IMPLEMENTATION OF CONTROLLER AREA NETWORK AND ITS APPLICATION

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

VLSI Design and Embedded System

By

TARANI CHAITANYA CHINTA

Roll No: 20507012

Department of Electronics & Communication Engineering

National Institute of Technology

Rourkela

2007

IMPLEMENTATION OF CONTROLLER AREA NETWORK AND ITS APPLICATION

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
VLSI Design and Embedded System

By

TARANI CHAITANYA CHINTA

Roll No: 20507012

Under the Guidance of

Prof. K. K. MAHAPATRA

Department of Electronics & Communication Engineering
National Institute of Technology
Rourkela

2007

NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA

CERTIFICATE

This is to certify that the thesis report titled "**Implementation of Controller Area Network and its application**" submitted by **Mr. Tarani Chaitanya Chinta** (Roll No: 20507012) in partial fulfillment of the requirements for the award of Master of Technology Degree in Electronics and Communication Engineering with specialization "**VLSI Design and Embedded System**" during session 2006-2007 at National Institute Of Technology, Rourkela (Deemed University) and is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/institute for the award of any Degree or Diploma.

Date:

Prof. K. K. Mahapatra
Department of E.C.E
National Institute of Technology
Rourkela-769008

ACKNOWLEDGEMENTS

It is a pleasure to thank many people who made this thesis possible.

I would like to take this opportunity to express my gratitude and sincere thanks to my supervisor **Prof. K. K. Mahapatra** for his guidance, insight, and support he has provided throughout the course of this work.

I am grateful to our teachers **Prof. G.S. Rath**, **Prof. G. Panda, Prof. S.K. Patra** and **Dr. S. Meher**. From these teachers I learned about the great role of self-learning and the constant drive for understanding emerging technologies, and a passion for knowledge.

My special thanks go to research scholars, friends and juniors at NIT Rourkela for their encouragement and help throughout the course.

I would like to thank all faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela for their extreme help throughout course.

Finally, I am forever indebted to my parents and sisters for their love, understanding, endless patience and encouragement when it was most required. I am also grateful to Suri's family for their affection and support.

Tarani Chaitanya Chinta

CONTENTS

ABSTRACT

The Controller Area Network (CAN) is a serial, asynchronous, multi-master communication protocol for connecting electronic control modules in automotive and industrial applications. CAN was designed for automotive applications needing high levels of data integrity and data rates of up to 1 Mbit/s. In this project Controller Area Network protocol is implemented using on chip Motorola Scalable Controller Area Network (MSCAN) of MC9S12DP256B 16-bit Microcontroller. The application we have taken up is “Monitoring of Temperature using LM35 based on Controller Area Network architecture” .The system is constituted of two CAN nodes, each CAN node is formed by a transceiver MC33388 and 16-bit microcontroller MC9S12DP256B.

The first stage consist of the conventional sensor of temperature (LM35) that converts the room temperature into voltage signal, and then this signal is conditioned and then the signal is transmitted to the input of the A/D converter of the microcontroller. The A/D converted data is transmitted to the CAN node 2 using the CAN architecture. At node 2 the received temperature readings are displayed using an alphanumeric LCD display of size 16X2. Node 2 consists of a keypad through which user can enter the maximum operating temperature of the device. According to the maximum temperature value entered the node 2 can take decisions and controls temperature source which is at node1 by sending control signals via the CAN network as per the user requirements.

In this thesis we have concerned with design techniques for implementation of CAN nodes for data monitoring and taking appropriate decision based on data in the control system. Implementation of CAN for temperature monitoring and controlling the device is successful and the same idea can be applicable to monitor tire pressure monitoring system, Adaptive Cruise control, power window and Engine management systems in Automotives. This leads to decentralization of control system in vehicles. This can be extended to industrial control applications.

List of Figures

Chapter 1

INTRODUCTION

An embedded system is a microprocessor based system that is built to control a function or range of functions and is not designed to be programmed by the end user in the same way that a PC is. With a PC, this is exactly what a user can do; one minute the PC is word processor and next it's a games machine simply by changing the software. An embedded system is designed to perform on particular task albeit with choices and different options. The last point is important because it differentiates itself from the world of the PC where the end user does reprogram it whenever a different software package is bought and run. However, PCs have provided an easily accessible source of hardware and software for embedded systems and it should be no surprise that they form the basis of many embedded systems.

1.1 Inside the embedded system

Processor: The main criterion for the processor is: can it provide the processing power needed to perform the tasks within the system. While processor performance is essential and forms the first gating criterion, there are others such as cost; this should be system cost and not just the cost of the processor in isolation, power consumption, software tools and component availability and so on.

Memory: Memory is an important part of any embedded system design and is heavily influenced by the software design, and in turn may dictate how the software is designed, written and developed. It provides storage for the software that it will run. With RAM being more expensive than ROM and non-volatile, many embedded systems and in particular, microcontrollers, have small amounts of RAM compared to the ROM that is available for the program.

Peripherals: An embedded system has to communicate with the outside world and this is done by peripherals. Input peripherals are usually associated with sensors that measure the external environment and thus effectively control the output operations that the embedded system performs. Our work is confined in this area of embedded systems. Specially, there are different data communication protocols used to transfer the data from one device to another device in the network of embedded systems and devices. In this thesis work we studied three different protocols and developed hardware and software for the protocols on microcontroller (Freescale 68HCS12) platform.

Software: The software component within an embedded system often encompasses the technology that adds value to the system and defines what it does and how well it does it.

Embedded software consists of different components: initialization and configuration, operating system or run time environment, application software, error handling and debug.

Algorithms: Algorithms are the key constituents of the software that makes an embedded system behave in the way that it does. They can range from mathematical processing through to models of the external environment which are used to interpret information from external sensors and thus generate control signals. With the digital technology in use today such as MP3 and DVD players, the algorithms that digitally encode the analogue data defined by standard bodies.

1.2 Networks in Embedded systems

There are several reasons to build network based embedded systems. When the processing tasks are physically distributed, it may be necessary to put some of the computing power near where the events occur. Data reduction is another important reason for distributed processing. It may be possible to perform some initial signal processing on captured data to reduce its volume. Reducing the data on separate processor may significantly reduce the load on the processor that makes use of that data. Modularity is another motivation for network based design. When a large system is assembled out of existing components, those components may use a network port as clean interface that does not interface with the internal operation of the component in ways that using the microprocessors in one part of the network can be used to probe components in another part of network. Distributed embedded system design is another example of hardware/software co- design, since we must design the network topology as well as the software running on the network nodes.

1.3 Necessity of Networks in Embedded Systems

Building an embedded system with several processing elements talking over a network is definitely more complicated than using single large microprocessors to perform the same tasks. Distributed systems are necessary because the devices that the processing elements communicate with are physically separated. If the deadlines for processing the data are short, it may be cost –effective to put the processing elements where the data are located rather than build a higher speed network to carry the data to distant, fast processing elements.

An important advantage of distributed system with several CPU's is that one part of the system can be used to help diagnose problems in another part. Whether you are debugging a prototype or diagnosing a problem in the field, isolating the error to one part of the system can be difficult when everything is done on single CPU. If you have several CPU's in the system, you can use one to generate inputs for another and to watch its output.

To optimize Input and Output ports of System on Chip (SoC) and microcontrollers is challenging task to Circuit designer. We can use several serial protocols where very high speed data transfer is not required. Few protocols will come to help of Circuit designer to optimize Input and Output ports.

Finally, a system may be required to be distributed if it makes use of component subsystems that have their own embedded microprocessors. The LCD display subsystem included a microcontroller that accepted data to be displayed and controlled the LCD display signals as required. The display inherently introduced distributed element into the architecture. Of course, the designers made other uses the distributed style. Some components were connected by serial bus, which augmented the system bus that connected the high speed processors with program/data memory.

1.4 Networks for Embedded systems

Many different networks have seen widespread use in distributed embedded systems over the years. Several system buses have been used to build distributed machines as well as for their original purpose of supporting multicard computer systems. Examples of these types are Multibus and VME developed by Intel and Motorola for multicard computer systems, PCI bus etc.

Several interconnect networks have been developed especially for distributed embedded computing. Few examples are:

Inter IC communication, popularly known as IIC developed by Philips Semiconductors (NXP Semiconductors) around 1986. This bus commonly used to link microcontrollers into systems. IIC is designed to be low cost, easy to implement and of moderate speed (100 kbps). It uses only two lines: serial data line (SDA) and serial clock line (SCL), which indicates when valid data are on the data line. IIC bus is designed as multimaster bus- any one of several different devices may act as the master at various times.

Serial Peripheral Interface (SPI) was developed by Motorola Semiconductors (Freescale Semiconductors). This bus is used to connect peripheral devices like LCD, memory to microprocessors. SPI uses four lines: MOSI (Master out and Slave in), MISO (master in and slave out), clock line and chip select line. This is also true multimaster bus.

The Controller Area Network (CAN) bus was developed for automotive electronics. It provides megabit rates and can handle large numbers of devices. The Echelon LON network was developed for home and industrial automation. Many DSP processors supply their own

interconnect structures for multiprocessing. In addition, many networks designed for general purpose computing have been put to use in embedded applications as well.

New microcontrollers are available in the market, which provides TCP/IP connectivity as peripheral and IP core has been implemented in chip, which can be used in web based control systems. Myrinet used in many high-performance signal processing systems.

We can find many protocols coming out from research. We should be careful while selecting these protocols to design embedded systems. We need to check out with system specifications, design requirements, availability of devices based on respective protocol and cost of system. Based on above conditions we can draw few concrete rules while selecting communication protocols:

1. Maximum and minimum speed of data transfer or baud rate of the communication protocol.
2. Data security or corruption of data due to electromagnetic interference (EMI) in the PCB or in the system. So protocol should be immune to EMI.
3. Size of data: chunk of data that can be transferred on the protocol for each frame. Protocols supports 8 bit, 16 bit and 32 bit data transfers
4. Distance between the nodes in the network. Efficient performance can be achieved by restricting the distance between nodes. For example I2C will give best performance if we restrict the distance between nodes is 10 to 12 feet.
5. Number of nodes or devices in the network matters when it is very complex systems like Automobiles or Space craft. Designer expects protocol standards should support more number of devices
6. Good error handling mechanisms.

Real time systems in automotives and real time systems need a protocol which supports unlimited number of nodes (some thousands of nodes), very high baud rate, efficient error handling system and distance between the nodes more than 100 feet. Inter IC communication or SPI does not satisfy above requirements. So under the leadership of BOSCH, Society of Automotive Engineers (SAE) was founded in 1994 to define the protocol standards especially for the Automotives. SAE came out with SAE J1850 BDLC (Byte data link communication) protocol in 1995, which satisfied the above requirements. In 1999, SAE came out with famous protocol called Controller Area Network (CAN), which is upgraded version of the J1850.

Data transfer through CAN was successful previously. Our contribution is, we have used CAN with 29 bit identifier, which leads to connecting the devices on the network, more than 11 bit identifier mode. CAN based networks used separate CAN controller and CAN transceiver connected to microcontroller. Here, we are using CAN IP of Freescale HCS12 microcontroller, this leads to reduction in PCB size and cost of production. HCS12 also consists of on chip Analog to Digital Converter, which leads to further reduction in hardware on board and cost.

Our work is implementation of sensors for monitoring the temperature and communicating among the network using the protocol CAN. The successful implementation of CAN leads to decentralization of control systems in Automotives and Industrial control systems. Devices in the network, which receives the temperature data, are capable of taking decisions. These decisions can include the control of actuator, the sending of information for a controller in communication to other devices that can carry out a wished function. CAN is protocol of serial communication that allows the control of systems distributed in real time with high level of safety. This protocol was developed in the early nineties in order to decentralize vehicles control systems because their characteristics CAN are being used currently in different areas like Automotives, Industrial Control and medical instrumentation.

Chapter 2

CONTROLLER AREA NETWORK

2.1 Introduction

The Controller Area Network (CAN) is a serial communications protocol that efficiently supports distributed real-time control applications with a very high level of data integrity. CAN was originally developed by the German company Robert Bosch for use in the car industry to provide a cost-effective communications bus for in-car electronics and as alternative to expensive and cumbersome wiring looms. The car industry continues to use CAN for an increasing number of applications, but because of its proven reliability and robustness, CAN is now also being used in many other industrial control applications. CAN fulfils the communication needs of a wide range of applications, from high-speed networks to low-cost multiplex wiring. For example, in automotive electronics, engine control units, sensors and anti-skid systems may be connected using CAN, with bit-rates up to 1 Mbit/s. At the same time, it is cost effective to build CAN into vehicle body electronics, such as lamp clusters and electric windows, to replace the wiring harness otherwise required. The intention of the CAN specification is to achieve compatibility between any two CAN implementations. Compatibility, however, has different aspects with respect to, for example, electrical features and the interpretation of data to be transferred.

To achieve design transparency and implementation flexibility CAN has been subdivided into different layers according to the ISO/OSI Reference Model:

• The Data Link Layer

  – The Logical Link Control (LLC) sublayer

  – The Medium Access Control (MAC) sublayer

• The Physical Layer

In previous versions of the CAN specification the services and functions of the LLC and MAC sublayers of the Data Link Layer were described as layers called Object Layer and Transfer Layer. The scope of the LLC sublayer is

• to provide services for data transfer and remote data request,

• to decide which messages received by the LLC sublayer are actually to be accepted

• to provide means for recovery management and overload notifications.

There is considerable freedom in defining object handling. The MAC sublayer mainly is the transfer protocol, i.e. controlling the framing, performing arbitration, error checking, error signalling and fault confinement. Within the MAC sublayer it is decided whether the bus is free for starting a new transmission, or whether a reception of a message is just starting. Also, some general features of the bit-timing are regarded as part of the MAC sublayer. Modifications to the MAC sublayer cannot be made.

The scope of the Physical Layer is the actual transfer of the bits between the different nodes, with respect to all electrical properties. Within one network the physical layer has to be the same for all nodes. However, there are many possible implementations of the Physical Layer.

2.2 Basic Concepts

Layered structure of a CAN node

The LLC sublayer is concerned with message filtering, overload notification and recovery management. The MAC sublayer represents the kernel of the CAN protocol. It presents messages received from the LLC sublayer and accepts messages to be transmitted by the LLC sublayer. The MAC sublayer is responsible for message framing, arbitration, acknowledgement, error detection and signalling. The MAC sublayer is supervised by a self checking mechanism, called fault confinement, which distinguishes short disturbances from permanent failures. The Physical Layer defines how signals are actually transmitted, dealing with the descriptions of bit timing, bit encoding and synchronization. The Physical Layer is not defined here, as it will vary according to the requirements of individual applications (for example, transmission medium and signal level implementations).

Messages

Information on the bus is sent in fixed format messages of different but limited length. When the bus is free, any connected node may start to transmit a new message.

Information routing

In CAN systems a node does not make use of any information about the system configuration (e.g. node addresses). This has several important consequences, which are described below.

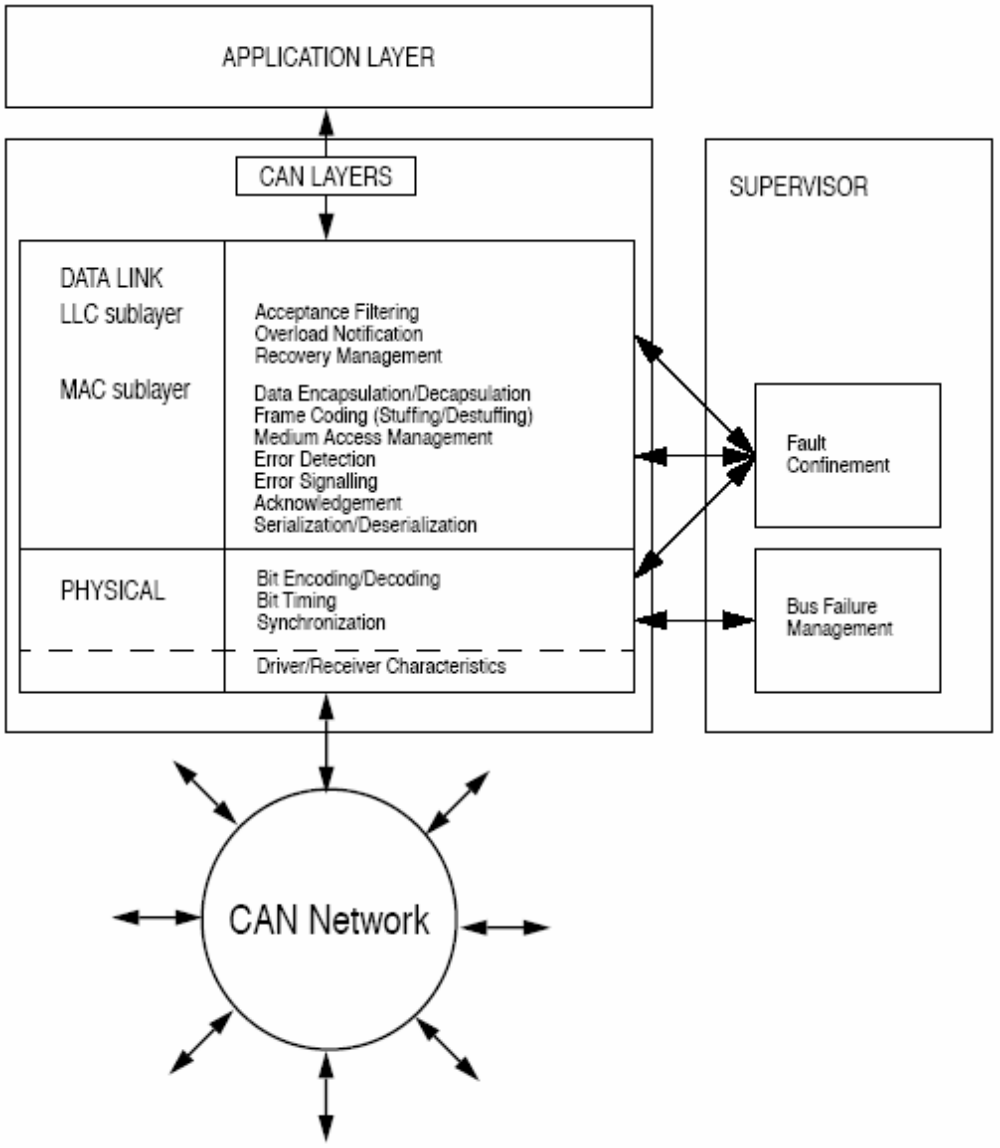


Fig 2.1: CAN Layers

System flexibility

Nodes may be added to the CAN network without requiring any change in the software or hardware of any node or the application layer.

Message routing

The content of a message is named by an Identifier. The Identifier does not indicate the destination of the message, but describes the meaning of the data, so that all nodes in the network are able to decide by Message Filtering whether the data is to be acted upon by them or not.

Multicast

As a consequence of the concept of Message Filtering any number of nodes may receive and simultaneously act upon the same message.

Data consistency

Within a CAN network it is guaranteed that a message is accepted simultaneously either by all nodes or by no node. Thus data consistency of a system is achieved by the concepts of multicast and by error handling.

Bit-rate

The speed of CAN may be different in different systems. However, in a given system the bit-rate is uniform and fixed.

Priorities

The Identifier defines a static message priority during bus access.

Remote data request

By sending a Remote frame a node requiring data may request another node to send the corresponding Data frame. The Data frame and the corresponding Remote frame have the same Identifier.

Multi-master

When the bus is free any node may start to transmit a message. The node with the message of highest priority to be transmitted gains bus access.

Arbitration

Whenever the bus is free, any node may start to transmit a message. If two or more nodes start transmitting messages at the same time, the bus access conflict is resolved by bit-wise arbitration using the Identifier. The mechanism of arbitration guarantees that neither information nor time is lost. If a Data frame and a Remote frame with the same Identifier are initiated at the same time, the Data frame prevails over the Remote frame. During arbitration every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal the node may continue to send. When a ‘recessive’ level is sent and a ‘dominant’ level is monitored, the node has lost arbitration and must withdraw without sending any further bits.

Data integrity

In order to achieve a very high integrity of data transfer, powerful measures of error detection, signalling and self-checking are implemented in every CAN node.

Error detection

To detect errors the following measures have been taken:

• Monitoring (each transmitter compares the bit levels detected on the bus with the bit levels being transmitted)

• Cyclic Redundancy Check (CRC)

• Bit-Stuffing

• Message Frame Check

Performance of error detection

The error detection mechanisms have the following properties:

• Monitoring:

– All global errors are detected

– All local errors at transmitters are detected

• CRC:

– Up to 5 randomly distributed errors within a message are detected

– Burst errors of length less than 15 in a message are detected

– Errors of any odd number of bits in a message are detected

The total residual error probability of undetected corrupted messages is less than message error rate **4.7×10^{-11}**.

Error signalling and recovery time

Corrupted messages are flagged by any node detecting an error. Such messages are aborted and will be retransmitted automatically. The recovery time from detecting an error until the start of the next message is at most 31 bit times, provided there is no further error.

Fault confinement

CAN nodes are able to distinguish short disturbances from permanent failures. Defective nodes are switched off.

Connections

The CAN serial communication link is a bus to which a number of nodes may be connected. This number has no theoretical limit. Practically, the total number of nodes will be limited by delay times and/or electrical loads on the bus line.

Single channel

The bus consists of a single bidirectional channel that carries bits. From this data resynchronization information can be derived. These channels can be implemented using single wire (plus ground), two differential wires, optical fibres, etc.

Bus values

The bus can have one of two complementary values: dominant or recessive. During simultaneous transmission of dominant and recessive bits, the resulting bus value will be dominant. For example, in the case of a wired-AND implementation of the bus, the dominant level would be represented by a logical ‘0’ and the recessive level by a logical ‘1’.

Acknowledgement

All receivers check the consistency of the message being received and will acknowledge a consistent message and flag an inconsistent message.

Sleep mode/wake-up

To reduce the system's power consumption, a CAN device may be set into sleep mode, in which there is no internal activity and the bus drivers are disconnected. The sleep mode is finished with a wake-up by any bus activity or by internal conditions of the system. On wake-up, the internal activity is restarted, although the MAC sublayer will wait for the system's oscillator to stabilize and then wait until it has synchronized itself to the bus activity (by checking for eleven consecutive recessive bits), before the bus drivers are set to the on-bus state again.

Oscillator Tolerance

A maximum oscillator tolerance of 1.58% is given allowing ceramic resonators to be used in applications with transmission rates of up to 125 kbps, as a general rule. A quartz oscillator is required to achieve the full bus speed range of the CAN protocol. The chip of the CAN network with the highest requirement for oscillator accuracy determines the oscillator accuracy from all the other nodes.

2.3 Message Transfer

2.3.1 Definition of transmitter/receiver

Transmitter

A node originating a message is called TRANSMITTER of that message. The node continues to be TRANSMITTER until the bus is idle or the node loses Arbitration.

Receiver

A node is called RECEIVER of a message if it is not the TRANSMITTER of that message, and the bus is not idle.

2.3.2 Frame formats

There are two different frame formats which differ from each other by the length of their Identifier fields. Frames with 11 bit Identifier fields are denoted Standard Frames, while frames with 29 bit Identifier fields are denoted Extended frames.

2.3.3 Frame types

Message transfer is manifested and controlled by four different frame types:

• A Data frame carries data from a transmitter to the receivers.

• A Remote frame is transmitted by a bus node to request the transmission of the Data frame with the same Identifier.

• An Error frame is transmitted by any node on detecting a bus error.

• An Overload frame is used to provide for an extra delay between the preceding and the succeeding Data or Remote frames. Data frames and Remote frames are separated from preceding frames by an Interframe space.

2.3.3.1 Data frame

A Data frame is composed of seven different bit fields:

Start of frame, Arbitration field, Control field, Data field, CRC field, ACK field, End of frame. The Data field can be of length zero.

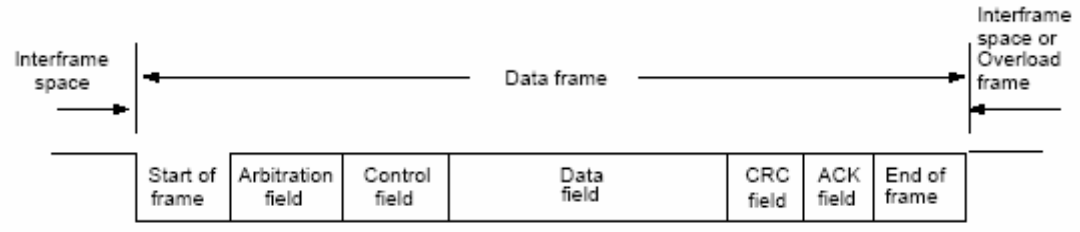


Fig 2.2: Data Frame

Start of frame

Start of frame marks the beginning of Data frames and Remote frames. It consists of a single dominant bit. A node is only allowed to start transmission when the bus is idle. All nodes have to synchronize to the leading edge caused by Start of frame of the node starting transmission first.

Arbitration field

The format of the Arbitration field is different for Standard Format and Extended Format frames.

• In Standard Format the Arbitration field consists of the 11 bit Identifier and the RTR-Bit. The Identifier bits are denoted ID-28 ... ID-18.

• In Extended Format the Arbitration field consists of the 29 bit Identifier, the SRR-Bit, the IDE-Bit, and the RTR-Bit. The Identifier bits are denoted ID-28 ... ID-0.

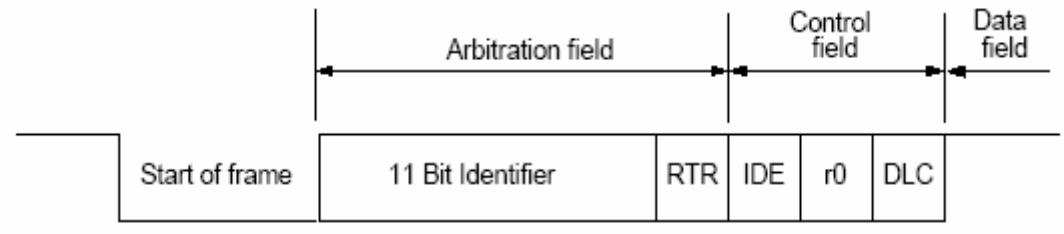


Fig 2.3: Arbitration Field; Standard Format

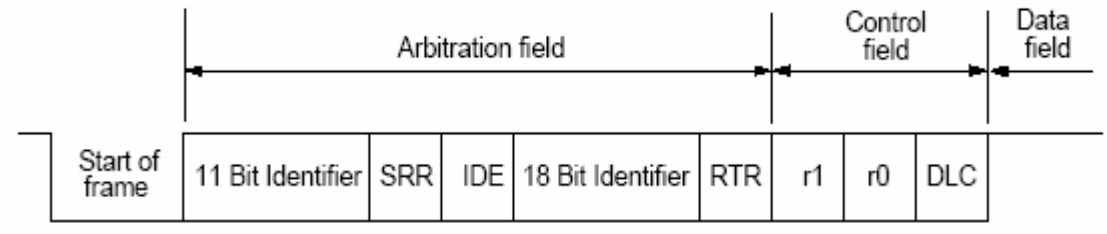


Fig 2.4: Arbitration Field; Extended Format

Identifier

In Standard Format the Identifier's length is 11 bits and corresponds to the Base ID in Extended Format. These bits are transmitted in the order from ID-28 to ID-18. The least significant bit is ID-18. The 7 most significant bits (ID-28 - ID-22) must not be all recessive.

In Extended Format the Identifier's length is 29 bits. The format comprises two sections; Base ID with 11 bits and the Extended ID with 18 bits.

• Base ID: The Base ID consists of 11 bits. It is transmitted in the order from ID-28 to ID-18 and is equivalent to the format of the Standard Identifier. The Base ID defines the Extended Frame's base priority.

• Extended ID: The Extended ID consists of 18 bits. It is transmitted in the order of ID-17 to ID-0.

In a Standard Frame the Identifier is followed by the RTR bit.

RTR BIT (Standard Format and Extended Format)

Remote Transmission Request bit.

In Data frames the RTR bit has to be dominant. Within a Remote frame the RTR bit has to be recessive.

In an Extended Frame the Base ID is transmitted first, followed by the IDE bit and the SRR bit. TheExtended ID is transmitted after the SRR bit.

SRR BIT (Extended Format)

Substitute Remote Request bit.

The SRR is a recessive bit. In Extended Frames the SRR bit is transmitted at the position of the RTR bit in Standard Frames and so substitutes for the RTR bit in the Standard Frame.

As a consequence, collisions between a Standard Frame and an Extended Frame, where the Base ID of both frames is the same, are resolved in such a way that the Standard Frame prevails over the Extended Frame.

IDE BIT (Extended Format)

Identifier Extension bit.

The IDE bit belongs to:

• The Arbitration field for the Extended Format

• The Control field for the Standard Format

The IDE bit in the Standard Format is transmitted dominant, whereas in the Extended Format the IDE bit is recessive.

Control field

The Control field consists of six bits. The format of the Control field is different for Standard Format and Extended Format. Frames in Standard Format include the Data length code, the IDE bit, which is transmitted dominant, and the reserved bit r0. Frames in Extended Format include the Data length code and two reserved bits, r0 and r1. The reserved bits must be sent dominant, but the receivers accept dominant and recessive bits in all combinations.

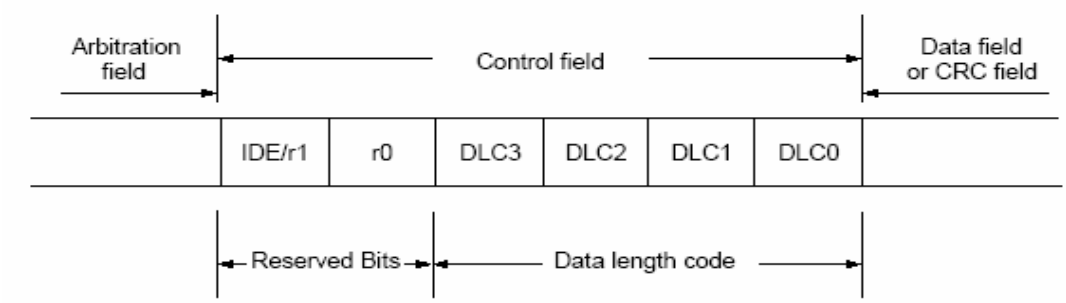


Fig 2.5 Control Field Standard Format and Extended Format

Data Length ode (Standard Format and Extended Format)

The number of bytes in the Data field is indicated by the Data length code. This Data length code is 4 bits wide and is transmitted within the Control field. The DLC bits can code data lengths from 0 to 8 bytes and other values are not permitted.

Data field

The Data field consists of the data to be transferred within a Data frame. It can contain from 0 to 8 bytes, each of which contain 8 bits which are transferred MSB first.

Table 2.1 Data length coding

DATA LENGTH CODE				DATA BYTE COUNT
DLC3	DLC2	DLC1	DLC0	
d	d	d	d	0
d	d	d	r	1
d	d	r	d	2
d	d	r	r	3
d	r	d	d	4
d	r	d	r	5
d	r	r	d	6
d	r	r	r	7
r	d	d	d	8
d = dominant r = recessive				

CRC field (Standard Format and Extended Format)

The CRC field contains the CRC Sequence followed by a CRC Delimiter.

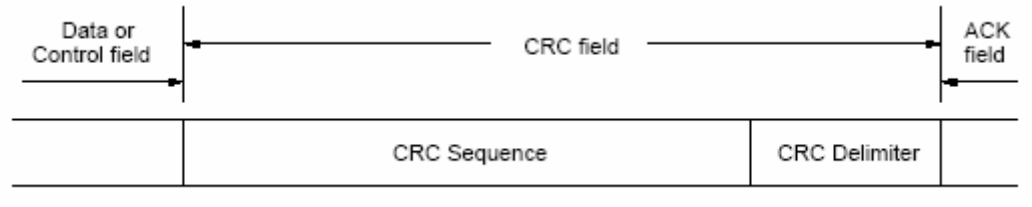


Fig 2.6: CRC Field

CRC Sequence

The frame check sequence is derived from a cyclic redundancy code best suited to frames with bit counts less than 127 bits (BCH Code). In order to carry out the CRC calculation the polynomial to be divided is defined as the polynomial whose coefficients are given by the destuffed bit-stream consisting of Start of frame, Arbitration field, Control field, Data field (if present) and, for the 15 lowest coefficients, by 0. This polynomial is divided by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1 \qquad (2.1)$$

The remainder of this polynomial division is the CRC Sequence transmitted over the bus.

CRC Delimiter (Standard Format and Extended Format**)**

The CRC Sequence is followed by the CRC Delimiter which consists of a single recessive bit.

ACK field (Standard Format and Extended Format)

The ACK field is two bits long and contains the ACK Slot and the ACK Delimiter. In the ACK field the transmitting node sends two recessive bits. A Receiver which has received a valid message correctly reports this to the Transmitter by sending a dominant bit during the ACK Slot (i.e. it sends ACK).

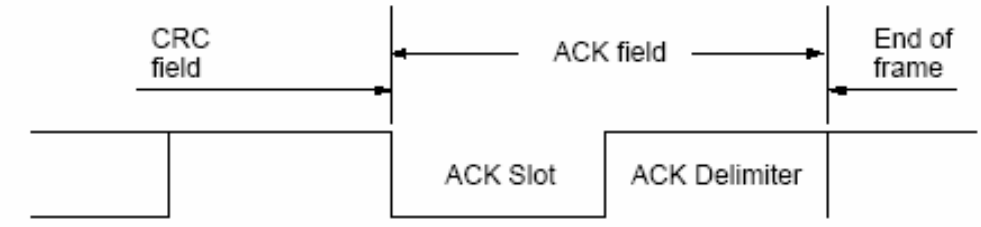


Fig 2.7: ACK Field

ACK Slot

All nodes having received the matching CRC Sequence report this within the ACK Slot by overwriting the recessive bit of the Transmitter by a dominant bit.

ACK Delimiter

The ACK Delimiter is the second bit of the ACK field and has to be a recessive bit. As a consequence, the ACK Slot is surrounded by two recessive bits (CRC Delimiter, ACK Delimiter).

End of frame

Each Data frame and Remote frame is delimited by a flag sequence consisting of seven recessive bits.

2.3.3.2 Remote frame

A node acting as a Receiver for certain data can initiate the transmission of the respective data by its source node by sending a Remote frame. A Remote frame is composed of six different bit fields: Start of frame, Arbitration field, Control field, CRC field, ACK field, End of frame. The RTR bit of a Remote frame is always recessive. There is no Data field in a Remote frame, irrespective of the value of the Data length code which is that of the corresponding Data frame and may be assigned any value within the admissible range 0... 8.

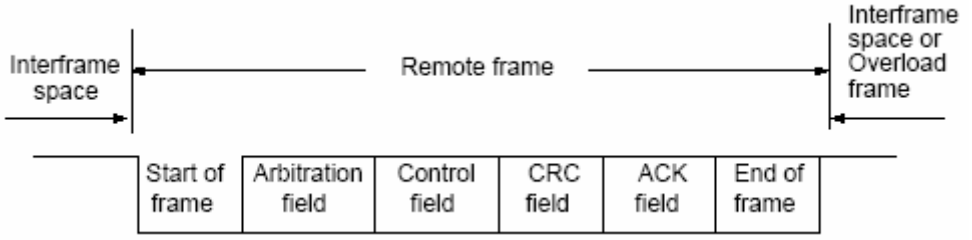


Fig 2.8: Remote Frame

The polarity of the RTR bit indicates whether a transmitted frame is a Data frame (RTR bit dominant) or a Remote frame (RTR bit recessive).

2.3.3.3 Error frame

The Error frame consists of two distinct fields. The first field is given by the superposition of Error flags contributed from different nodes. The second field is the Error Delimiter.

In order to terminate an Error frame correctly, an error-passive node may need the bus to be bus-idle for at least three bit times. Therefore the bus should not be loaded to 100%.

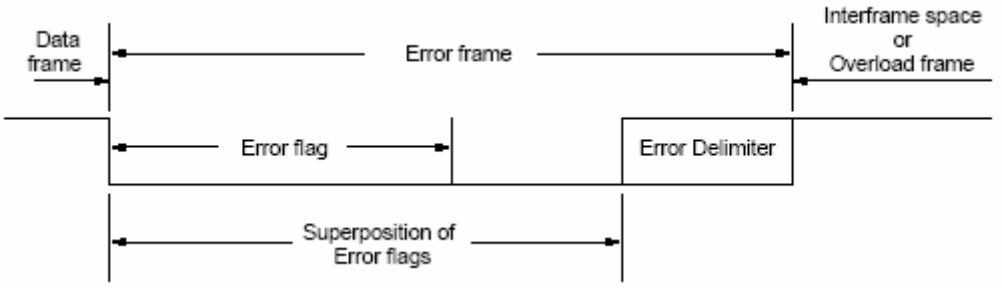


Fig 2.9: Error Frame

Error flag

There are two forms of error flag: an Active Error flag and a Passive Error flag.

1) Active Error flag consists of six consecutive dominant bits.

2) The Passive Error flag consists of six consecutive recessive bits unless it is overwritten by dominant bits from other nodes.

An error-active node detecting an error condition signals this by the transmission of an Active Error flag. The Error flag's form violates the law of bit stuffing, applied to all fields from Start of frame to CRC Delimiter or destroys the fixed form ACK field or End of frame field. As a consequence, all other nodes detect an error condition and each starts to transmit an Error flag. So the sequence of dominant bits which actually can be monitored on the bus results from a superposition of different Error flags transmitted by individual nodes. The total length of this sequence varies between a minimum of six and a maximum of twelve bits.

An error-passive node detecting an error condition tries to signal this by transmitting a Passive Error flag. The error-passive node waits for six consecutive bits of equal polarity, beginning at the start of the Passive Error flag. The Passive Error flag is complete when these six equal bits have been detected.

Error delimiter

The Error Delimiter consists of eight recessive bits. After transmission of an Error flag each node sends recessive bits and monitors the bus until it detects a recessive bit. Afterwards it starts transmitting seven more recessive bits.

2.3.3.4 Overload frame

The Overload frame contains two bit fields, Overload flag and Overload Delimiter. There are three kinds of Overload condition which lead to the transmission of an Overload flag:

1) Where the internal conditions of a receiver are such that the receiver requires a delay of the next Data frame or Remote frame.

2) Detection of a dominant bit at the first and second bit of Intermission.

3) If a CAN node samples a dominant bit at the eighth bit (i.e. the last bit) of an Error Delimiter or Overload Delimiter, it will start transmitting an Overload frame (not an Error frame). The Error Counters will not be incremented.

An Overload frame resulting from Overload condition 1 is only allowed to start at the first bit time of an expected Intermission, whereas Overload frames resulting from Overload conditions 2 and 3 start one bit after detecting the dominant bit.

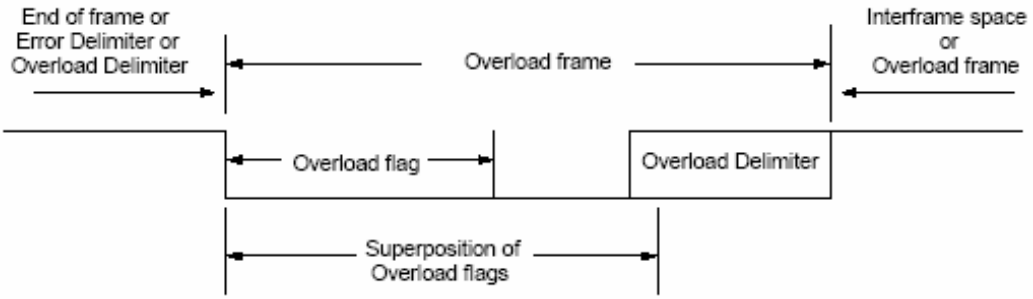


Fig 2.10: Over Load Frame

At most, two Overload frames may be generated to delay the next Data frame or Remote frame.

Overload flag

The Overload flag consists of six dominant bits. The overall form corresponds to that of the ACTIVE Error flag.

The Overload flag's form destroys the fixed form of the Intermission field. As a consequence, all other nodes also detect an Overload condition and each starts to transmit an Overload flag. In the event that there is a dominant bit detected during the third bit of Intermission then it will interpret this bit as Start of frame.

Overload delimiter

The Overload Delimiter consists of eight recessive bits.

The Overload Delimiter is of the same form as the Error Delimiter. After transmission of an Overload flag the node monitors the bus until it detects a transition from a dominant to a recessive bit. At this point of time every bus node has finished sending its Overload flag and all nodes start transmission of seven more recessive bits in coincidence.

2.3.3.5 Interframe space

Data frames and Remote frames are separated from preceding frames, whatever type they may be (Data frame, Remote frame, Error frame, Overload frame) by a field called Interframe

space. In contrast, Overload frames and Error frames are not preceded by an Interframe space and multiple Overload frames are not separated by an Interframe space.

The Interframe space contains the bit fields Intermission and Bus idle and, for error- passive nodes, which have been transmitter of the previous message, Suspend transmission.

(1) For nodes which are not error-passive or have been a receiver of the previous message:

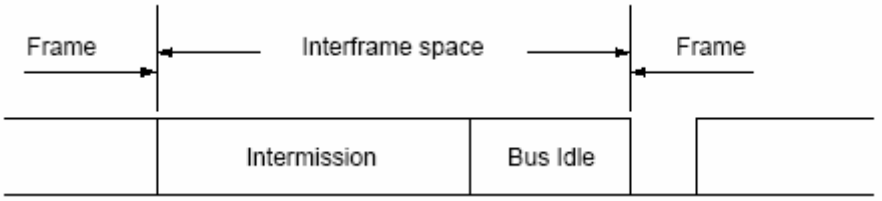


Fig2.11: Interframe space (1)

(2) For error-passive nodes which have been the transmitter of the previous message:

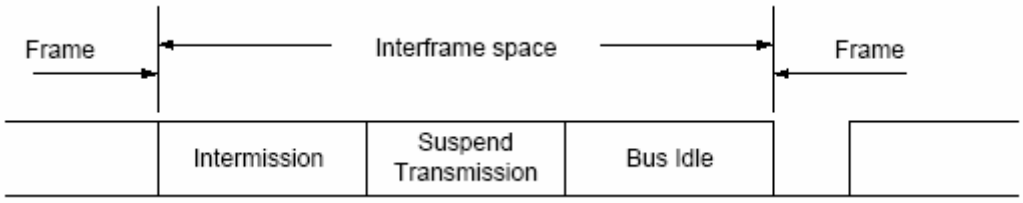


Fig2.12: Interframe space (2)

Intermission

Intermission consists of three recessive bits.

During Intermission no node is allowed to start transmission of a Data frame or Remote frame. The only action permitted is signalling of an Overload condition.

Bus idle

The period of Bus idle may be of arbitrary length. The bus is recognized to be free, and any node having something to transmit can access the bus. A message, pending during the transmission of another message, is started in the first bit following Intermission.

The detection of a dominant bit on the bus is interpreted as Start of frame.

Suspend transmission

After an error-passive node has transmitted a frame, it sends eight recessive bits following Intermission, before starting to transmit a further message or recognizing the bus to be idle. If meanwhile a transmission (caused by another node) starts, the node will become the receiver of this message.

2.3.4 Conformance with regard to frame formats

The Extended Format is a new feature of the CAN protocol. In order to allow the design of relatively simple controllers, the implementation of the Extended Format to its full extend is not required (e.g. send messages or accept data from messages in Extended Format), whereas the Standard Format must be supported without restriction.

New controllers are considered to be in conformance with this CAN Specification, if they have at least the following properties with respect to the Frame Formats defined in previous Section.

• Every new controller supports the Standard Format

• Every new controller can receive messages of the Extended Format. This requires that Extended frames are not destroyed just because of their format. It is, however, not required that the Extended Format must be supported by new controllers.

2.4 Message filtering

Message filtering is based upon the whole Identifier. Optional mask registers that allow any Identifier bit to be set 'don't care' for message filtering, may be used to select groups of Identifiers to be mapped into the attached receive buffers.

If mask registers are implemented every bit of the mask registers must be programmable, i.e. they can be enabled or disabled for message filtering. The length of the mask register can comprise the whole Identifier or only part of it.

2.5 CAN Frame Format-Standard Format

Fig 2.13: Data Frame

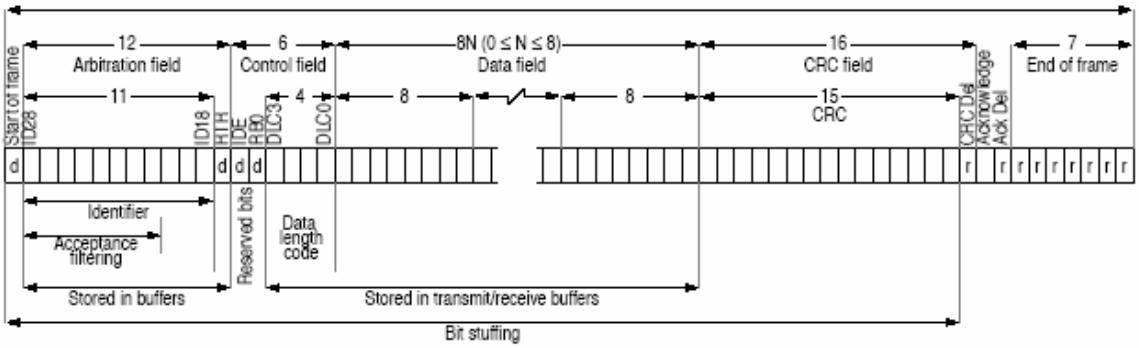


Fig 2.14: Remote Frame

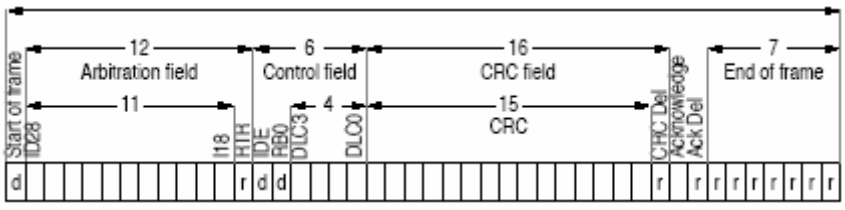


Fig 2.15: Error Frame

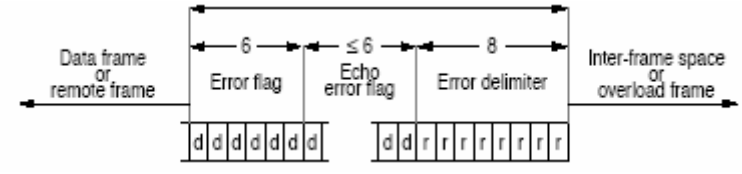


Fig 2.16: Inter Frame Space

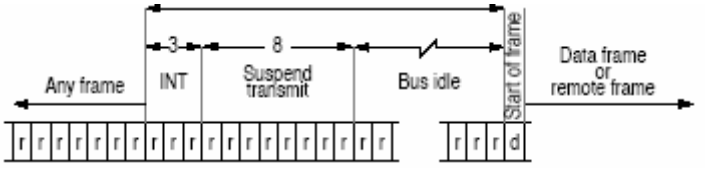
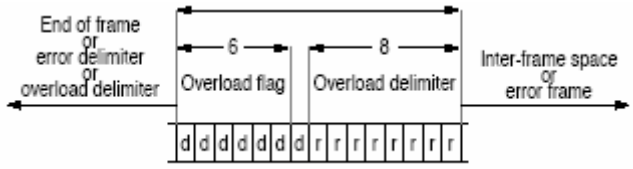


Fig 2.17: Overload Frame



2.6 CAN Frame Format-Extended Format

Fig 2.18: Data Frame

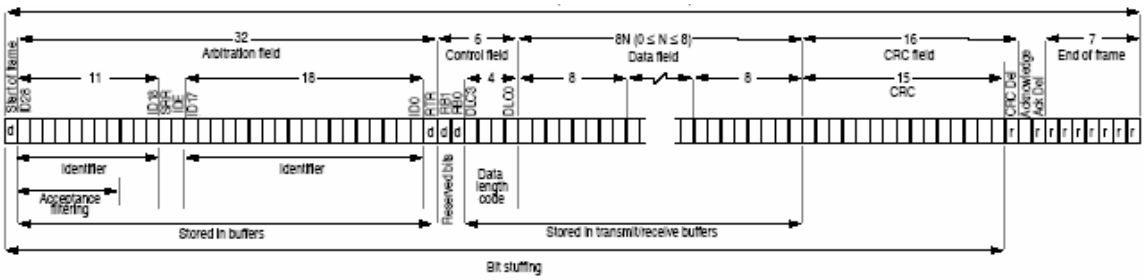
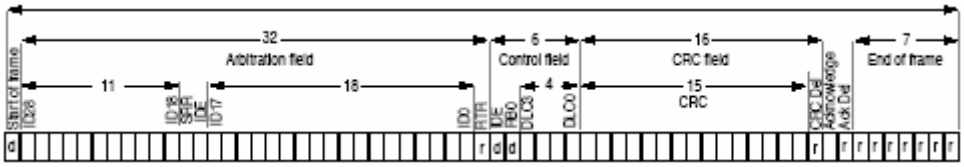


Fig 2.19: Remote Frame



2.7 Message validation

The point of time at which a message is taken to be valid is different for the transmitter and the receivers of the message.

2.7.1 Transmitter

The message is valid for the transmitter if there is no error until the End of frame. If a message is corrupted, retransmission will follow automatically and according to the rules of

prioritization. In order to be able to compete for bus access with other messages, retransmission has to start as soon as the bus is idle.

2.7.2 Receiver

The message is valid for the receiver if there is no error until the last but one bit of End of frame.

2.8 Bit-stream coding

The frame segments Start of frame, Arbitration field, Control field, Data field and CRC Sequence are coded by the method of bit stuffing. Whenever a transmitter detects five consecutive bits of identical value in the bit-stream to be transmitted it automatically inserts a complementary bit in the actual transmitted bit-stream.

The remaining bit fields of the Data frame or Remote frame (CRC Delimiter, ACK field and End of frame) are of fixed form and not stuffed. The Error frame and the Overload frame are also of fixed form and are not coded by the method of bit stuffing.

The bit-stream in a message is coded according to the Non-Return-to-Zero (NRZ) method. This means that during the total bit time the generated bit level is either dominant or recessive.

2.9 Error Handling

2.9.1 Error detection

There are five different error types. The following sections describe these errors.

Bit error

A node which is sending a bit on the bus also monitors the bus. A bit error has to be detected at that bit time, when the bit value that is monitored is different from the bit value that is sent. An exception to this is the sending of a recessive bit during the stuffed bit-stream of the Arbitration Field or during the ACK Slot. Then no Bit error occurs when a dominant bit is monitored. A transmitter sending a Passive error Flag and detecting a dominant bit does not interpret this as a Bit Error.

Stuff error

A Stuff Error has to be detected at the bit time of the 6^{th} consecutive equal bit level in a message field that should be coded by the method of bit stuffing.

CRC error

The CRC sequence consists of the result of the CRC calculation by the transmitter. The receivers calculate the CRC in the same way as the transmitter. A CRC Error must be recognized if the calculated result is not the same as that received in the CRC sequence.

Form error

A Form Error must be detected when a fixed-form bit field contains one or more illegal bits.

Acknowledgement error

An Acknowledgement Error must be detected by a transmitter whenever it does not monitor a dominant bit during ACK Slot.

2.9.2 Error signalling

A node detecting an error condition signals this by transmitting an Error Flag. An error-active node will transmit an ACTIVE Error Flag; an error-passive node will transmit a PASSIVE Error Flag. Whenever a Bit Error, a Stuff Error, a Form Error or an Acknowledgement Error is detected by any node, that node will start transmission of an Error Flag at the next bit time.

Whenever a CRC Error is detected, transmission of an Error Flag will start at the bit following the ACK Delimiter, unless an Error Flag for another error condition has already been started.

2.10 Fault Confinement

CAN node status

With respect to fault confinement, a node may be in one of three states:

- error-active
- error-passive
- bus off

An error active node can normally take part in bus communication and sends an Active Error Flag when an error has been detected.

An error-passive node must not send an Active Error Flag. It takes part in bus communication, but when an error has been detected only a Passive Error Flag is sent. Also after a transmission, an error-passive node will wait before initiating a further transmission.

A bus-off node is not allowed to have any influence on the bus (e.g. output drivers switched off).

Error counts

To facilitate fault confinement two counts are implemented in every bus node:

• Transmit error count

• Receive error count

These counts are modified according to the following rules:

1) When a Receiver detects an error, the Receive Error Count will be increased by 1, except when the detected error was a Bit Error during the sending of an Active Error Flag or an Overload Flag.

2) When a Receiver detects a dominant bit as the first bit after sending an Error Flag the Receive Error Count will be increased by 8.

3) When a Transmitter sends an Error Flag the Transmit Error Count is increased by 8.

Exception 1:

The Transmit Error Count is not changed if:

-The TRANSMITTER is error-passive and

– The TRANSMITTER detects an Acknowledgement Error because of not detecting a dominant ACK and

– The TRANSMITTER does not detect a dominant bit while sending its PASSIVE Error Flag

Exception 2:

The Transmit Error Count is not changed if:

– The TRANSMITTER sends an Error Flag because a Stuff Error occurred during Arbitration and

– The Stuff bit should have been recessive and

– The Stuff Bit has been sent as recessive but is monitored as dominant

4) An error-active Transmitter detects a Bit Error while sending an Active Error Flag or an Overload Flag, the Transmit Error Count is increased by 8.

5) An error-active Receiver detects a bit error while sending an Active Error Flag or an Overload Flag, the Receive Error Count is increased by 8.

6) Any node tolerates up to 7 consecutive dominant bits after sending an Active Error Flag, Passive Error Flag or Overload Flag. After detecting the eighth consecutive dominant bit and after each sequence of additional eight consecutive dominant bits, every Transmitter increases its Transmit Error Count by 8 and every Receiver increases its Receive Error Count by 8.

7) After the successful transmission of a message (getting ACK and no error until End of Frame is finished), the Transmit Error Count is decreased by 1, unless it was already 0.

8) After the successful reception of a message (reception without error up to the ACK Slot and the successful sending of the ACK bit), the Receive Error Count is decreased by 1, if it was between 1 and 127. If the Receive Error Count was 0, it stays 0, and if it was greater than 127, then it will be set to a value between 119 and 127.

9) A node is error passive when the Transmit Error Count equals or exceeds 128, or when the Receive Error Count equals or exceeds 128. An error condition letting a node become error-passive causes the node to send an Active Error Flag.

10) A node is bus-off when the Transmit Error Count is greater than or equal to 256.

11) An error-passive node becomes error-active again when both the Transmit Error Count and the Receive Error Count are less than or equal to 127.

12) A node which is bus-off is permitted to become error-active (no longer bus-off) with its error counters both set to 0 after 128 occurrences of 11 consecutive recessive bits have been monitored on the bus.

2.11 Bit Timing Requirements

Nominal bit rate

The Nominal bit rate is the number of bits per second transmitted in the absence of resynchronization by an ideal transmitter.

Nominal bit time

NOMINAL BIT TIME=1/NOMINAL BIT RATE

The Nominal bit time can be thought of as being divided into separate non overlapping time segments. These segments are as shown below, and form the bit time as shown in Figure

• SYNCHRONIZATION SEGMENT (SYNC_SEG)

• PROPAGATION TIME SEGMENT (PROP_SEG)

• PHASE BUFFER SEGMENT1 (PHASE_SEG1)

• PHASE BUFFER SEGMENT2 (PHASE_SEG2)

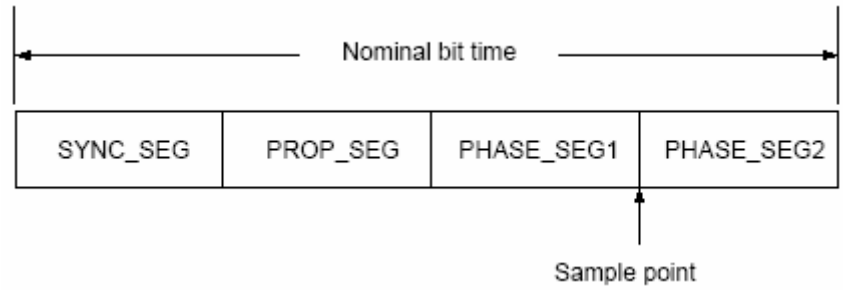


Fig 2.20: Partition of the Bit Time

SYNC_SEG

This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment

PROP_SEG

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay.

PHASE_SEG1, PHASE_SEG2

These Phase-Buffer-Segments are used to compensate for edge phase errors. These segments can be lengthened or shortened by resynchronization.

Sample point

The Sample point is the point in time at which the bus level is read and interpreted as the value of that respective bit. Its location is at the end of PHASE_SEG1.

Information processing time

The Information processing time is the time segment starting with the Sample point reserved for calculation of the subsequent bit level.

Time quantum

The Time quantum is a fixed unit of time which can be derived from the oscillator period. There is a programmable prescaler, with integral values (with a range of at least 1 to 32) which allows a fixed unit of time, the Time quantum can have a length of

$$\text{TIME QUANTUM} = m * \text{MINIMUM TIME QUANTUM} \quad (2.2)$$

Where m is the value of the prescaler.

Length of time segments

• SYNC_SEG is 1 Time quantum long

• PROP_SEG is programmable to be 1, 2 ...8 Time quanta long

• PHASE_SEG1 is programmable to be 1, 2 ...8 Time quanta long

• PHASE_SEG2 is the maximum of PHASE_SEG1 and the Information processing time

• The Information processing time is less than or equal to 2 Time quanta long

The total number of Time quanta in a bit time has to be programmable at least 8 to 25.

Fig 2.21: Bit timing of CAN devices without local CPU

Synchronization

Hard synchronization

After a Hard synchronization the internal bit time is restarted with SYNC_SEG. Thus Hard synchronization forces the edge which has caused the Hard synchronization to lie within the Synchronization segment of the restarted bit time.

Resynchronization jump width

As a result of resynchronization, PHASE_SEG1 may be lengthened or PHASE_SEG2 may be shortened. The amount of lengthening or shortening of the Phase buffer segments has an upper bound given by the Resynchronization jump width. The Resynchronization jump width is programmable between 1 and min (4, PHASE_SEG1).

Clocking information may be derived from transitions from one bit value to the other. The property that only a fixed maximum number of successive bits have the same value provides the possibility of resynchronising a bus node to the bit-stream during a frame. The maximum length between two transitions which can be used for resynchronization is 29 bit times.

Phase error of an edge

The Phase error of an edge is given by the position of the edge relative to SYNC_SEG, measured in Time quanta. The sign of Phase error is defined as follows:

- $e < 0$ if the edge lies after the Sample point of the previous bit.
- $e = 0$ if the edge lies within SYNC_SEG.
- $e > 0$ if the edge lies before the Sample point.

Resynchronization

The effect of a Resynchronization is the same as that of Hard synchronization, when the magnitude of the Phase error of the edge which causes the Resynchronization is less than or equal to the programmed value of the Resynchronization jump width. When the magnitude of the Phase error is larger than the Resynchronization jump width,

- and if the Phase error is positive, then PHASE_SEG1 is lengthened by an amount equal to the Resynchronization jump width.
- and if the Phase error is negative, then PHASE_SEG2 is shortened by an amount equal to the Resynchronization jump width.

Synchronization rules

Hard synchronization and Resynchronization are the two forms of Synchronization. They obey the following rules:

1) Only one Synchronization within one bit time is allowed.

2) An edge will be used for Synchronization only if the value detected at the previous Sample point (previous read bus value) differs from the bus value immediately after the edge.

3) Hard synchronization is performed whenever there is a recessive to dominant edge during Bus idle.

4) All other recessive to dominant edges fulfilling the rules 1 and 2 will be used for Resynchronization with the exception that a transmitter will not perform a Resynchronization as a result of a recessive to dominant edge with a positive Phase error, if only recessive to dominant edges are used for Resynchronization.

Chapter 3

MC9S12DP256B

MICROCONTROLLER

3.1 Overview

The MC9S12DP256 microcontroller unit (MCU) is a 16-bit device composed of standard on-chip peripherals including a 16-bit central processing unit (HCS12 CPU), 256K bytes of Flash EEPROM, 12K bytes of RAM, 4K bytes of EEPROM, two asynchronous serial communications interfaces (SCI), three serial peripheral interfaces (SPI), an 8-channel IC/OC enhanced capture timer, two 8-channel, 10-bit analog-to-digital converters (ADC), an 8-channel pulse-width modulator (PWM), a digital Byte Data Link Controller (BDLC), 29 discrete digital I/O channels (Port A, Port B, Port K and Port E), 20 discrete digital I/O lines with interrupt and wakeup capability, five CAN 2.0 A, B software compatible modules (MSCAN12), and an Inter-IC Bus. The MC9S12DP256 has full 16-bit data paths throughout. However, the external bus can operate in an 8-bit narrow mode so single 8-bit wide memory can be interfaced for lower cost systems. The inclusion of a PLL circuit allows power consumption and performance to be adjusted to suit operational requirements.

3.2 Features

• HCS12 Core

– 16-bit HCS12 CPU

i. Upward compatible with M68HC11 instruction set

ii. Interrupt stacking and programmer's model identical to M68HC11

iii. Instruction queue

iv. Enhanced indexed addressing

– MEBI (Multiplexed External Bus Interface)

– MMC (Module Mapping Control)

– INT (Interrupt control)

– BKP (Breakpoints)

– BDM (Background Debug Mode)

• CRG (low current oscillator, PLL, reset, clocks, COP watchdog, real time interrupt, clock monitor)

• 8-bit and 4-bit ports with interrupt functionality

– Digital filtering

– Programmable rising or falling edge trigger

• Memory

– 256K Flash EEPROM

– 4K byte EEPROM

– 12K byte RAM

• Two 8-channel Analog-to-Digital Converters

– 10-bit resolution

– External conversion trigger capability

• Five 1M bit per second, CAN 2.0 A, B software compatible modules

– Five receive and three transmit buffers

– Flexible identifier filter programmable as 2 x 32 bit, 4 x 16 bit or 8 x 8 bit

– Four separate interrupt channels for Rx, Tx, error and wake-up

– Low-pass filter wake-up function

– Loop-back for self test operation

• Enhanced Capture Timer

– 16-bit main counter with 7-bit prescaler

– 8 programmable input capture or output compare channels

– Two 8-bit or one 16-bit pulse accumulators

• 8 PWM channels

– Programmable period and duty cycle

– 8-bit 8-channel or 16-bit 4-channel

– Separate control for each pulse width and duty cycle

– Center-aligned or left-aligned outputs

– Programmable clock select logic with a wide range of frequencies

– Fast emergency shutdown input

– Usable as interrupt inputs

• Serial interfaces

– Two asynchronous Serial Communications Interfaces (SCI)

– Three Synchronous Serial Peripheral Interface (SPI)

• Byte Data Link Controller (BDLC)

– SAE J1850 Class B Data Communications Network Interface Compatible and ISO Compatible for Low-Speed (<125 Kbps) Serial Data Communications in Automotive Applications

• Inter-IC Bus (IIC)

– Compatible with I2C Bus standard

– Multi-master operation

– Software programmable for one of 256 different serial clock frequencies

• 112-Pin LQFP package

– I/O lines with 5V input and drive capability

– 5V A/D converter inputs

– Operation at 50MHz equivalent to 25MHz Bus Speed

– Development support

– Single-wire background debug™ mode (BDM)

– On-chip hardware breakpoints

3.3 Modes of Operation

User modes

• Normal and Emulation Operating Modes

– Normal Single-Chip Mode

– Normal Expanded Wide Mode

– Normal Expanded Narrow Mode

– Emulation Expanded Wide Mode

– Emulation Expanded Narrow Mode

• Special Operating Modes

– Special Single-Chip Mode with active Background Debug Mode

– Special Test Mode

– Special Peripheral Mode

Low power modes

• Stop Mode

• Pseudo Stop Mode

• Wait Mode

3.4 Block Diagram

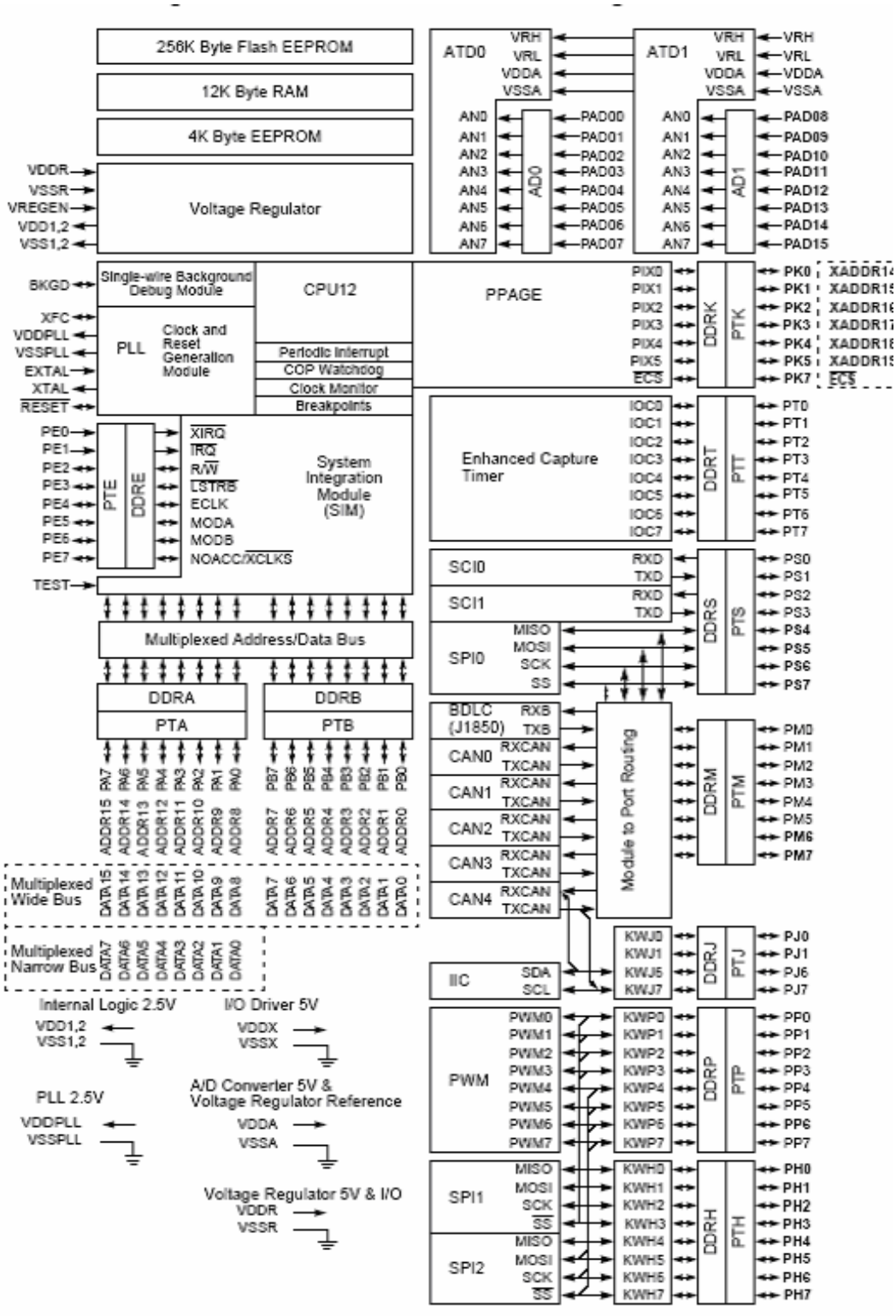


Fig 3.1: MC9S12DP256B Block Diagram

3.5 Device Memory Map

The following table and the figure show the device memory map of the MC9S12DP256B after reset. Note that after reset the bottom 1k of the EEPROM ($0000 - $03FF) are hidden by the register space.

Device Memory Map

Address	Module	Size (Bytes)
$0000 - $0017	CORE (Ports A, B, E, Modes, Inits, Test)	24
$0018 - $0019	Reserved	2
$001A - $001B	Device ID register (PARTID)	2
$001C - $001F	CORE (MEMSIZ, IRQ, HPRIO)	4
$0020 - $0027	Reserved	8
$0028 - $002F	CORE (Background Debug Mode)	8
$0030 - $0033	CORE (PPAGE, Port K)	4
$0034 - $003F	Clock and Reset Generator (PLL, RTI, COP)	12
$0040 - $007F	Enhanced Capture Timer 16-bit 8 channels	64
$0080 - $009F	Analog to Digital Converter 10-bit 8 channels (ATD0)	32
$00A0 - $00C7	Pulse Width Modulator 8-bit 8 channels (PWM)	40
$00C8 - $00CF	Serial Communications Interface 0 (SCI0)	8
$00D0 - $00D7	Serial Communications Interface 0 (SCI1)	8
$00D8 - $00DF	Serial Peripheral Interface (SPI0)	8
$00E0 - $00E7	Inter IC Bus	8
$00E8 - $00EF	Byte Data Link Controller (BDLC)	8
$00F0 - $00F7	Serial Peripheral Interface (SPI1)	8
$00F8 - $00FF	Serial Peripheral Interface (SPI2)	8
$0100- $010F	Flash Control Register	16
$0110 - $011B	EEPROM Control Register	12
$011C - $011F	Reserved	4
$0120 - $013F	Analog to Digital Converter 10-bit 8 channels (ATD1)	32
$0140 - $017F	Motorola Scalable Can (CAN0)	64
$0180 - $01BF	Motorola Scalable Can (CAN1)	64
$01C0 - $01FF	Motorola Scalable Can (CAN2)	64
$0200 - $023F	Motorola Scalable Can (CAN3)	64
$0240 - $027F	Port Integration Module (PIM)	64
$0280 - $02BF	Motorola Scalable Can (CAN4)	64
$02C0 - $03FF	Reserved	320
$0000 - $0FFF	EEPROM array	4096
$1000 - $3FFF	RAM array	12288
$4000 - $7FFF	Fixed Flash EEPROM	16384
$8000 - $BFFF	Flash EEPROM Page Window	16384
$C000 - $FFFF	Fixed Flash EEPROM	16384

Fig 3.2 Device Memory Map

MC9S12DP256B Memory Map

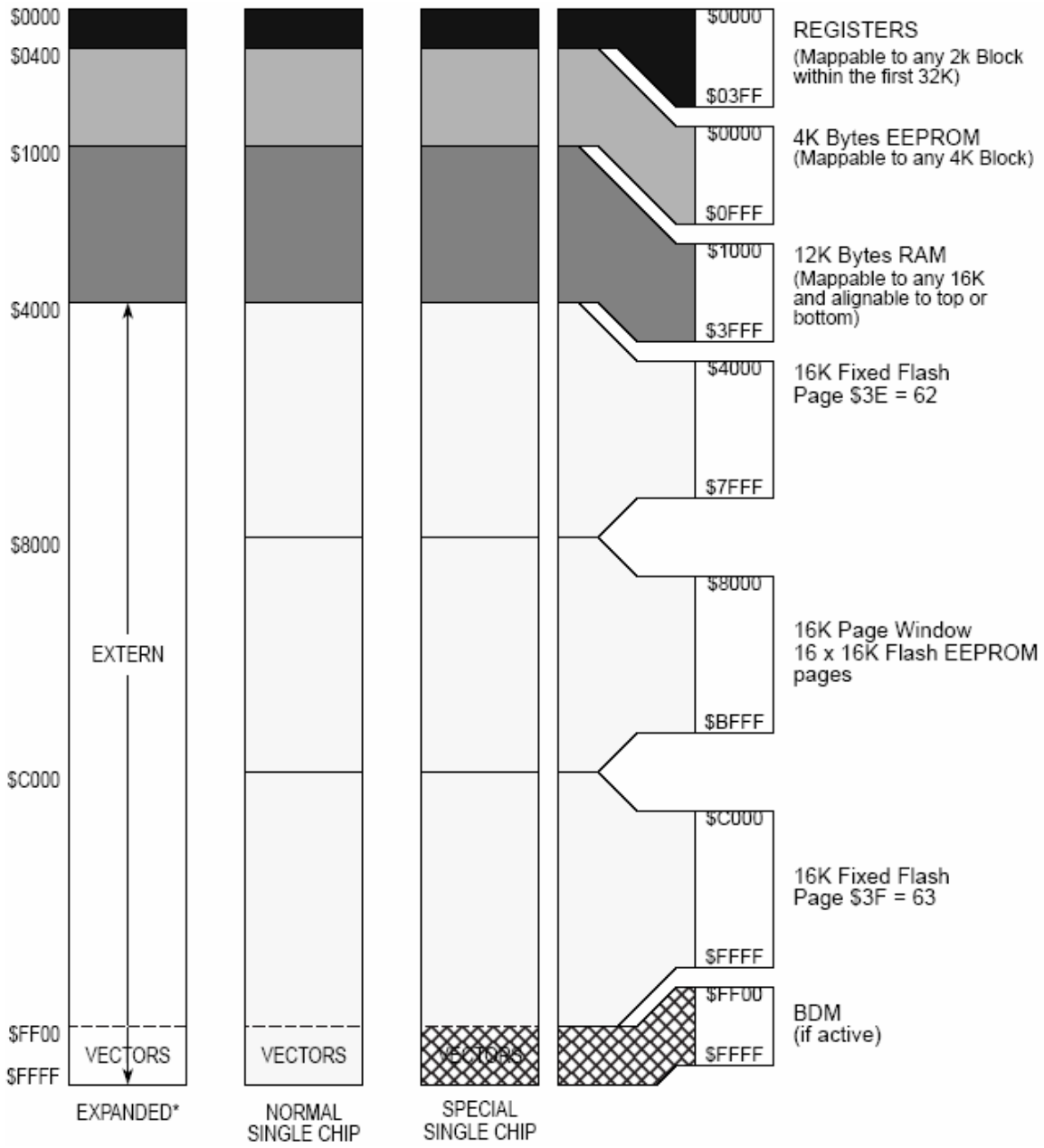


Fig 3.3: MC9S12DP256B Memory Map

3.6 Ports:

Port A: In port A the pins PA7-PA0 are general purpose input or output pins. In MCU expanded modes of operation, these pins are used for the multiplexed external address and data bus.

Port B: Port B is a general purpose input or output port same as Port A. In MCU expanded modes of operation, this port is used for the multiplexed external address and data bus.

Port E: It is a general purpose input or output port.

PE7 / NOACC / XCLKS — Port E I/O Pin 7

PE7 is a general purpose input or output pin. During MCU expanded modes of operation, the NOACC signal, when enabled, is used to indicate that the current bus cycle is an unused or "free" cycle. This signal will assert when the CPU is not using the bus. The XCLKS input selects between an external clock or oscillator configuration. The state of this pin is latched at the rising edge of RESET. If the input is a logic low the EXTAL pin is configured for an external clock drive. If input is a logic high an oscillator circuit is configured on EXTAL and XTAL. Since this pin is an input with a pull-up device, if the pin is left floating, the default configuration is an oscillator circuit on EXTAL and XTAL.

PE6 / MODB / IPIPE1 — Port E I/O Pin 6

PE6 is a general purpose input or output pin. It is used as a MCU operating mode select pin during reset. The state of this pin is latched to the MODB bit at the rising edge of RESET. This pin is shared with the instruction queue tracking signal IPIPE1. This pin is an input with a pull-down device which is only active when RESET is low.

PE5 / MODA / IPIPE0 — Port E I/O Pin 5

PE5 is a general purpose input or output pin. It is used as a MCU operating mode select pin during reset. The state of this pin is latched to the MODA bit at the rising edge of RESET. This pin is shared with the instruction queue tracking signal IPIPE0. This pin is an input with a pull-down device which is only active when RESET is low.

PE4 / ECLK — Port E I/O Pin 4

PE4 is a general purpose input or output pin. It can be configured to drive the internal bus clock ECLK. ECLK can be used as a timing reference.

PE3 / LSTRB / TAGLO — Port E I/O Pin 3

PE3 is a general purpose input or output pin. In MCU expanded modes of operation, LSTRB can be used for the low-byte strobe function to indicate the type of bus access and when instruction tagging is on, TAGLO is used to tag the low half of the instruction word being read into the instruction queue.

PE2 / R/W—Port E I/O Pin 2

PE2 is a general purpose input or output pin. In MCU expanded modes of operations, this pin drives the read/write output signal for the external bus. It indicates the direction of data on the external bus.

PE1 / IRQ — Port E Input Pin 1

PE1 is a general purpose input pin and the maskable interrupt request input that provides a means of applying asynchronous interrupt requests. This will wake up the MCU from STOP or WAIT mode.

PE0 / XIRQ — Port E Input Pin 0

PE0 is a general purpose input pin and the non-maskable interrupt request input that provides a means of applying asynchronous interrupt requests. This will wake up the MCU from STOP or WAIT mode.

Port H: It is a general purpose input or output port.

PH7 / KWH7 / SS2 — Port H I/O Pin 7

PH7 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as slave select pin SS of the Serial Peripheral Interface 2 (SPI2).

PH6 / KWH6 / SCK2 — Port H I/O Pin 6

PH6 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 2 (SPI2).

PH5 / KWH5 / MOSI2 — Port H I/O Pin 5

PH5 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 2 (SPI2).

PH4 / KWH4 / MISO2 — Port H I/O Pin 2

PH4 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 2 (SPI2).

PH3 / KWH3 / SS1 — Port H I/O Pin 3

PH3 is a general purpose input or output pin. It can be configured to generate an interrupt causing the CU o exit STOP or WAIT mode. It can be configured as slave select pin SS of the Serial Peripheral Interface (SPI1).

PH2 / KWH2 / SCK1 — Port H I/O Pin 2

PH2 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 1 (SPI1).

PH1 / KWH1 / MOSI1 — Port H I/O Pin 1

PH1 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 1 (SPI1).

PH0 / KWH0 / MISO1 — Port H I/O Pin 0

PH0 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 1 (SPI1).

Port J: It is a general purpose input output port.

PJ7 / KWJ7 / TXCAN4 / SCL — PORT J I/O Pin 7

PJ7 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as the transmit pin TXCAN for the Motorola Scalable Controller Area Network controller 4 (CAN4) or the serial clock pin SCL of the IIC module.

PJ6 / KWJ6 / RXCAN4 / SDA — PORT J I/O Pin 6

PJ6 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as the receive pin RXCAN for the Motorola Scalable Controller Area Network controller 4 (CAN4) or the serial data pin SDA of the IIC module.

PJ [1:0] / KWJ [1:0] — Port J I/O Pins [1:0]

PJ1 and PJ0 are general purpose input or output pins. They can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode.

Port K

PK7 / ECS / ROMONE — Port K I/O Pin 7

PK7 is a general purpose input or output pin. During MCU expanded modes of operation, this pin is used as the emulation chip select output (ECS). During MCU normal expanded wide and narrow modes of operation, this pin is used to enable the Flash EEPROM memory in the

memory map (ROMONE). At the rising edge of RESET, the state of this pin is latched to the ROMON bit.

PK [5:0] / XADDR [19:14] — Port K I/O Pins [5:0]

PK5-PK0 are general purpose input or output pins. In MCU expanded modes of operation, these pins provide the expanded address XADDR [19:14] for the external bus.

Port M

PM7 / TXCAN3 / TXCAN4 — Port M I/O Pin 7

PM7 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controllers 3 or 4 (CAN3 or CAN4).

PM6 / RXCAN3 / RXCAN4 — Port M I/O Pin 6

PM6 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controllers 3 or 4 (CAN3 or CAN4).

PM5 / TXCAN2 / TXCAN0 / TXCAN4 / SCK0 — Port M I/O Pin 5

PM5 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controllers 2, 0 or 4 (CAN2, CAN0 or CAN4). It can be configured as the serial clock pin SCK of the Serial Peripheral Interface 0 (SPI0).

PM4 / RXCAN2 / RXCAN0 / RXCAN4/ MOSI0 — Port M I/O Pin 4

PM4 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controllers 2, 0 or 4 (CAN2, CAN0 or CAN4). It can be configured as the master output (during master mode) or slave input pin (during slave mode) MOSI for the Serial Peripheral Interface 0 (SPI0).

PM3 / TXCAN1 / TXCAN0 / SS0 — Port M I/O Pin 3

PM3 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controllers 1 or 0 (CAN1 or CAN0). It can be configured as the slave select pin SS of the Serial Peripheral Interface 0 (SPI0).

PM2 / RXCAN1 / RXCAN0 / MISO0 — Port M I/O Pin 2

PM2 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controllers 1 or 0 (CAN1 or CAN0). It can be configured as the master input (during master mode) or slave output pin (during slave mode) MISO for the Serial Peripheral Interface 0 (SPI0).

PM1 / TXCAN0 / TXB — Port M I/O Pin 1

PM1 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controller 0 (CAN0). It can be configured as the transmit pin TXB of the BDLC.

PM0 / RXCAN0 / RXB — Port M I/O Pin 0

PM0 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controller 0 (CAN0). It can be configured as the receive pin RXB of the BDLC.

Port P

PP7 / KWP7 / PWM7 / SCK2 — Port P I/O Pin 7

PP7 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 7 output. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 2 (SPI2).

PP6 / KWP6 / PWM6 / SS2 — Port P I/O Pin 6

PP6 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 6 output. It can be configured as slave select pin SS of the Serial Peripheral Interface 2 (SPI2).

PP5 / KWP5 / PWM5 / MOSI2 — Port P I/O Pin 5

PP5 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 5 output. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 2 (SPI2).

PP4 / KWP4 / PWM4 / MISO2 — Port P I/O Pin 4

PP4 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 4 output. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 2 (SPI2).

PP3 / KWP3 / PWM3 / SS1 — Port P I/O Pin 3

PP3 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width

Modulator (PWM) channel 3 output. It can be configured as slave select pin SS of the Serial Peripheral Interface 1 (SPI1).

PP2 / KWP2 / PWM2 / SCK1 — Port P I/O Pin 2

PP2 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 2 output. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 1 (SPI1).

PP1 / KWP1 / PWM1 / MOSI1 — Port P I/O Pin 1

PP1 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 1 output. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 1 (SPI1).

PP0 / KWP0 / PWM0 / MISO1 — Port P I/O Pin 0

PP0 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 0 output. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 1 (SPI1).

Port S

PS7 / SS0 — Port S I/O Pin 7

PS6 is a general purpose input or output pin. It can be configured as the slave select pin SS of the Serial Peripheral Interface 0 (SPI0).

PS6 / SCK0 — Port S I/O Pin 6

PS6 is a general purpose input or output pin. It can be configured as the serial clock pin SCK of the Serial Peripheral Interface 0 (SPI0).

PS5 / MOSI0 — Port S I/O Pin 5

PS5 is a general purpose input or output pin. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 0 (SPI0).

PS4 / MISO0 — Port S I/O Pin 4

PS4 is a general purpose input or output pin. It can be configured as master input (during master mode) or slave output pin (during slave mode) MOSI of the Serial Peripheral Interface 0 (SPI0).

PS3 / TXD1 — Port S I/O Pin 3

PS3 is a general purpose input or output pin. It can be configured as the transmit pin TXD of Serial Communication Interface 1 (SCI1).

PS2 / RXD1 — Port S I/O Pin 2

PS2 is a general purpose input or output pin. It can be configured as the receive pin RXD of Serial Communication Interface 1 (SCI1).

PS1 / TXD0 — Port S I/O Pin 1

PS1 is a general purpose input or output pin. It can be configured as the transmit pin TXD of Serial Communication Interface 0 (SCI0).

PS0 / RXD0 — Port S I/O Pin 0

PS0 is a general purpose input or output pin. It can be configured as the receive pin RXD of Serial Communication Interface 0 (SCI0).

Port Ad

PAD15 / AN15 / ETRIG1 — Port AD Input Pin of ATD1

PAD15 is a general purpose input pin and analog input AN7 of the analog to digital converter ATD1. It can act as an external trigger input for the ATD1.

PAD[14:08] / AN[14:08] — Port AD Input Pins of ATD1

PAD14 - PAD08 are general purpose input pins and analog inputs AN[6:0] of the analog to digital converter ATD1.

PAD7 / AN07 / ETRIG0 — Port AD Input Pin of ATD0

PAD7 is a general purpose input pin and analog input AN7 of the analog to digital converter ATD0. It can act as an external trigger input for the ATD0.

PAD[06:00] / AN[06:00] — Port AD Input Pins of ATD0

PAD06 - PAD00 are general purpose input pins and analog inputs AN[6:0] of the analog to digital converter ATD0.

Port T

PT[7:0] / IOC[7:0] — Port T I/O Pins [7:0]

PT7-PT0 are general purpose input or output pins. They can be configured as input capture or output compare pins IOC7-IOC0 of the Enhanced Capture Timer (ECT).

3.7 Enhanced Capture Timer

3.7.1Overview

The HCS12 Enhanced Capture Timer module has the features of the HCS12 Standard Timer module enhanced by additional features in order to enlarge the field of applications, in particular for automotive ABS applications. This design specification describes the standard

timer as well as the additional features. The basic timer consists of a 16-bit, software-programmable counter driven by a prescaler. This timer can be used for many purposes, including input waveform measurements while simultaneously generating an output waveform. Pulse widths can vary from microseconds to many seconds. A full access for the counter registers or the input capture/output compare registers should take place in one clock cycle. Accessing high byte and low byte separately for all of these registers may not yield the same result as accessing them in one word.

3.7.2 Features

• 16-Bit Buffer Register for four Input Capture (IC) channels.

• Four 8-Bit Pulse Accumulators with 8-bit buffer registers associated with the four buffered IC channels. Configurable also as two 16-Bit Pulse Accumulators.

• 16-Bit Modulus Down-Counter with 4-bit Prescaler.

• Four user selectable Delay Counters for input noise immunity increase.

3.7.3 Block diagram

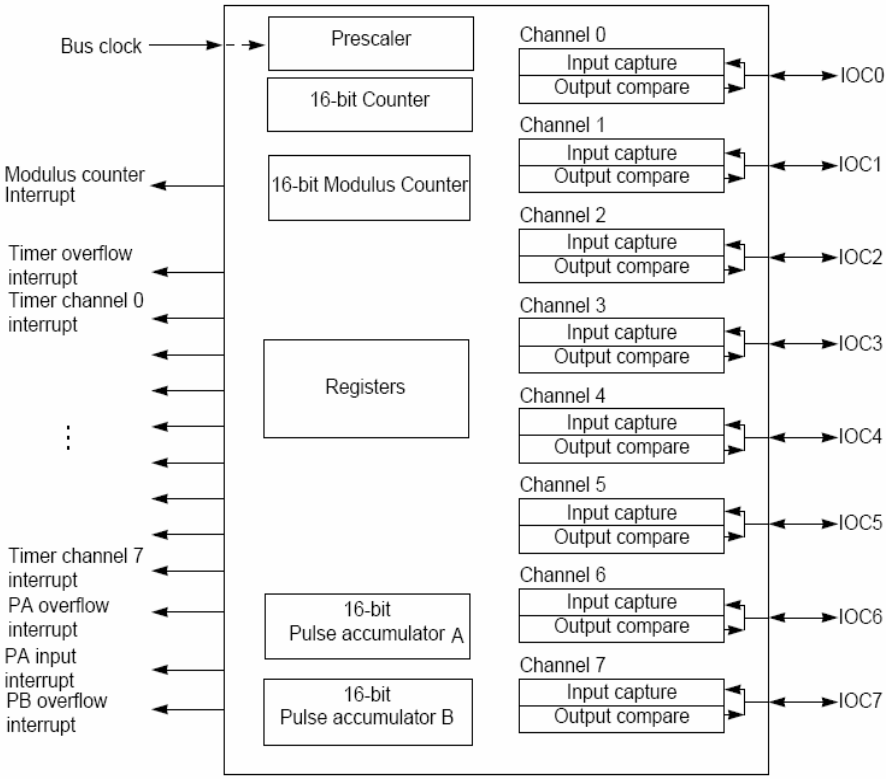


Fig 3.4 Enhanced Capture Timer Block Diagram

3.7.4 ECT Modes of operation

The Enhanced Capture Timer has 8 Input Capture, Output Compare (IC/OC) channels same as on theHC12 standard timer (timer channels TC0 to TC7). When channels are selected as input capture by selecting the IOSn bit in TIOS register, they are called Input Capture (IC) channels. Four IC channels are the same as on the standard timer with one capture register

each which memorizes the timer value captured by an action on the associated input pin. Four other IC channels, in addition to the capture register, have also one buffer each called holding register. This permits to memorize two different timer values without generation of any interrupt. Four 8-bit pulse accumulators are associated with the four buffered IC channels. Each pulse accumulator has a holding register to memorize their value by an action on its external input. Each pair of pulse accumulators can be used as a 16-bit pulse accumulator. The 16-bit modulus down-counter can control the transfer of the IC registers contents and the pulse accumulators to the respective holding registers for a given period, every time the count reaches zero. The modulus down-counter can also be used as a stand-alone time base with periodic interrupt capability.

The basic timer consists of a 16-bit software-programmable up-counter, driven by a pre-scaled clock signal. In normal mode the timer keeps running until it is switched off explicitly. There also exists a modulus down counter. This type of counter decrements an associated counter register until it reaches 0x0000 (underflow), whereupon the initial starting value is reloaded. The modulus down counter is therefore well-suited to the generation of fixed-period time base signals.

3.8 Analog to Digital converter

3.8.1 Overview

The Analog to Digital converter is an 8-channel, 10-bit, multiplexed input successive approximation analog-to-digital converter. The block is designed to be upwards compatible with the 68HC11 standard 8-bit A/D converter.

3.8.2 Features

• 8/10 Bit Resolution.
• 7μsec, 10-Bit Single Conversion Time.
• Sample Buffer Amplifier.
• Programmable Sample Time.
• Left/Right Justified, Signed/Unsigned Result Data.
• External Trigger Control.
• Conversion Completion Interrupt Generation.
• Analog Input Multiplexer for 8 Analog Input Channels.
• Analog/Digital Input Pin Multiplexing.
• 1 to 8 Conversion Sequence Lengths.
• Continuous Conversion Mode.
• Multiple Channel Scans.

3.8.3 ATD Block Diagram

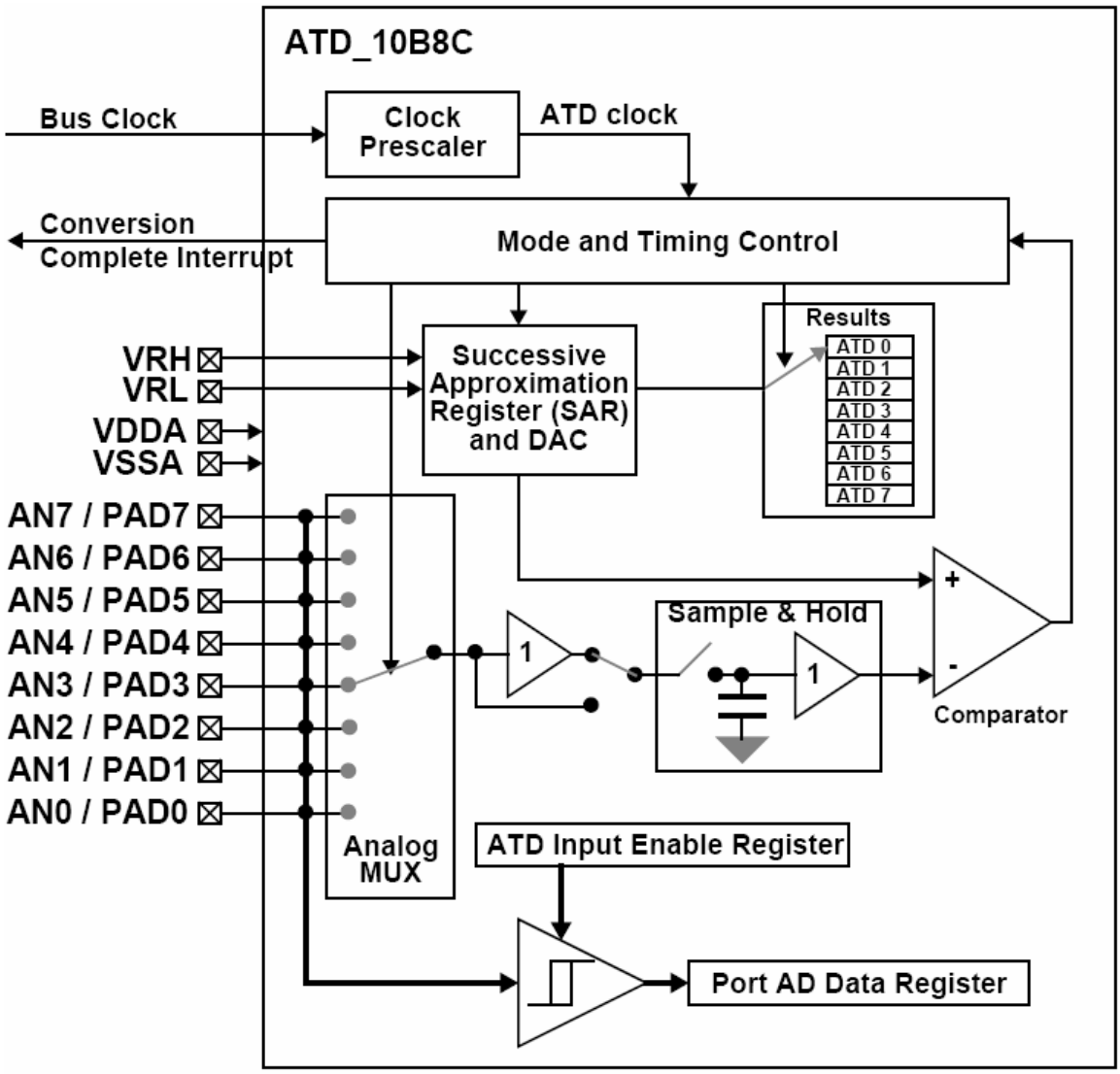


Fig 3.5: ATD Block Diagram

3.8.4 ATD Machine

The A/D Machine performs analog to digital conversions. The resolution is program selectable at either 8 or 10 bits. The A/D machine uses a successive approximation architecture. It functions by comparing the stored analog sample potential with a series of digitally generated analog potentials. By following a binary search algorithm, the A/D machine locates the approximating potential that is nearest to the sampled potential. When not converting the A/D machine disables its own clocks. The analog electronics still draws quiescent current. The power down (ADPU) bit must be set to disable both the digital clocks and the analog power consumption. Only analog input signals within the potential range of V_{RL} to V_{RH} (A/D reference potentials) will result in a non-railed digital output codes.

3.8.5 General Purpose Digital Input Port Operation

The input channel pins can be multiplexed between analog and digital data. As analog inputs, they are multiplexed and sampled to supply signals to the A/D converter. As digital inputs, they supply external input data that can be accessed through the digital port register PORTAD (input-only). The analog/digital multiplex operation is performed in the input pads. The input pad is always connected to the analog inputs of the ATD converter. The input pad signal is buffered to the digital port registers. This buffer can be turned on or off with the ATDDIEN register. This is important so that the buffer does not draw excess current when analog potentials are presented at its input.

3.9 Inter Integrated Circuit Bus

3.9.1 Overview

The Inter-IC Bus (IIC or I2C) is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange between devices. Being a two-wire device, the IIC Bus minimizes the need for large numbers of connections between devices, and eliminates the need for an address decoder.

This bus is suitable for applications requiring occasional communications over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface will operate at baud rates of up to 100kbps with maximum capacitive bus loading. With reduced bus slew rate, the device is capable of operating at higher baud rates, up to a maximum of [MCUbus] clock/20. The module can operate up to a baud rate of 400kbps provided the IIC bus slew rate is less than 100ns. The maximum communication interconnect length and the number of devices that can be connected to the bus are limited by a maximum bus capacitance of 400pF in all instances.

3.9.2 Features

The IIC module has the following key features:

• Compatible with IIC Bus standard

• Multi-master operation

• Software programmable for one of 256 different serial clock frequencies

• Software selectable acknowledge bit

• Interrupt driven byte-by-byte data transfer

• Arbitration lost interrupt with automatic mode switching from master to slave

• Calling address identification interrupt

• Start and stop signal generation/detection

• Repeated start signal generation

• Acknowledge bit generation/detection

•Bus busy detection

3.9.3 The block diagram of IIC module

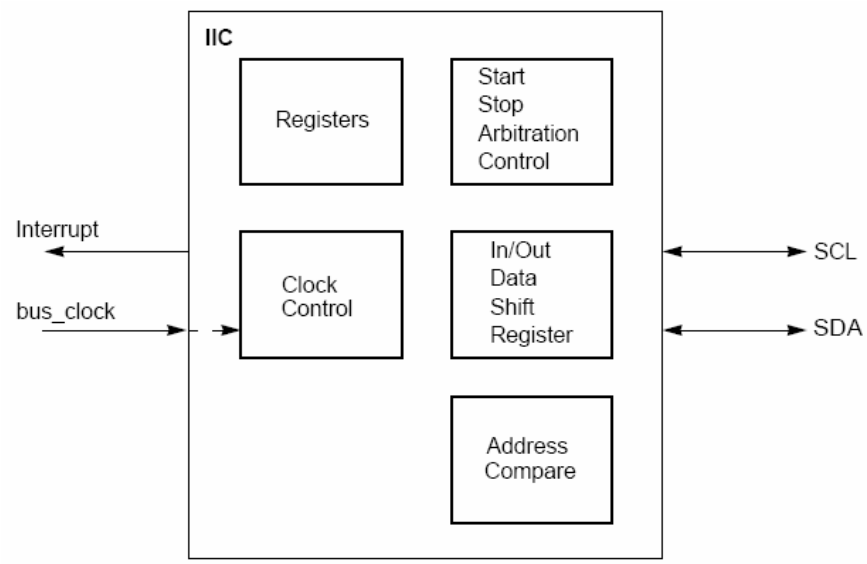


Fig 3.6: The block diagram of IIC module

The IIC Bus system uses a Serial Data line (SDA) and a Serial Clock Line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. Logic AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent. Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer and STOP signal.

3.10 Serial Communication Interface (SCI)

3.10.1 Overview

The SCI allows full duplex, asynchronous, serial communication between the CPU and remote devices, including other CPUs. The SCI transmitter and receiver operate independently, although they use the same baud rate generator. The CPU monitors the status of the SCI, writes the data to be transmitted, and processes received data.

3.10.2 Features

The SCI includes these distinctive features:

• Full-duplex operation

• Standard mark/space non-return-to-zero (NRZ) format

• 13-bit baud rate selection

• Programmable 8-bit or 9-bit data format

• Separately enabled transmitter and receiver

• Programmable transmitter output parity

• Two receiver wake up methods:

  – Idle line walk-up

  – Address mark walk-up

• Interrupt-driven operation with eight flags:

  – Transmitter empty

  – Transmission complete

  – Receiver full

  – Idle receiver input

  – Receiver overrun

  – Noise error

  – Framing error

  – Parity error

• Receiver framing error detection

• Hardware parity checking

• 1/16 bit-time noise detection

3.10.3 SCI Block Diagram

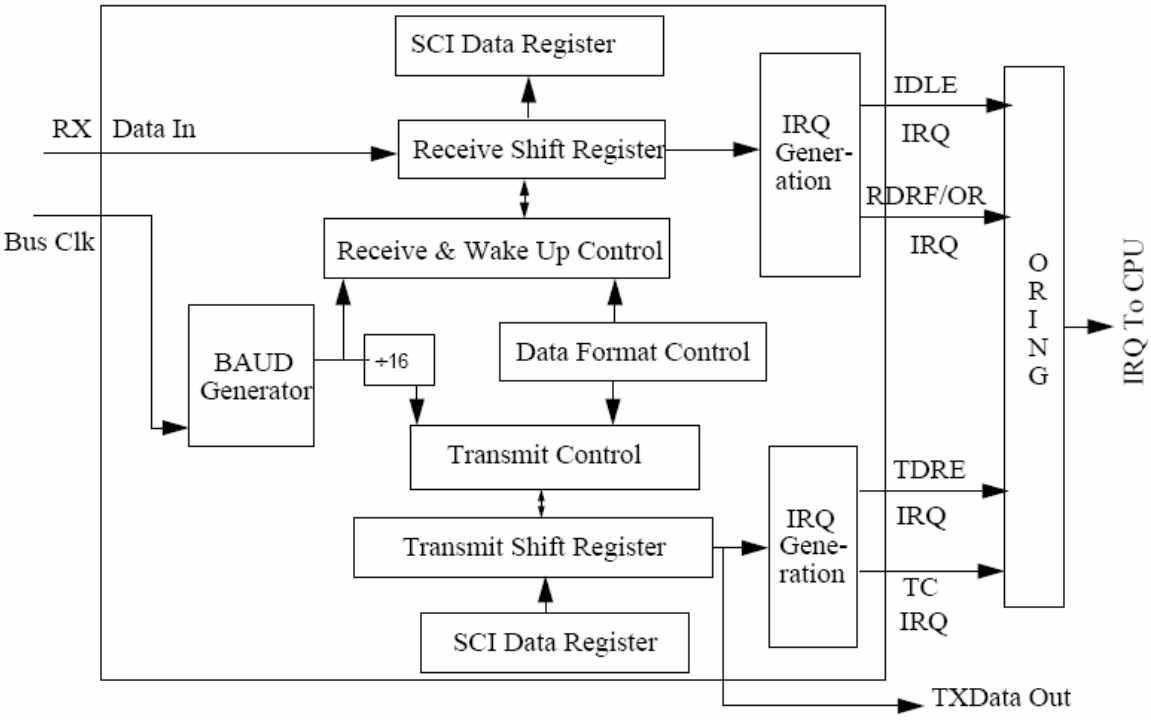


Fig 3.7: SCI Block diagram

3.11 Serial Peripheral Interface

3.11.1 Overview

The SPI module allows a duplex, synchronous, serial communication between the MCU and peripheral devices. Software can poll the SPI status flags or the SPI operation can be interrupt driven.

3.11.2 Features

The SPI includes these distinctive features:

• Master mode and slave mode

• Bi-directional mode

• Slave select output

• Mode fault error flag with CPU interrupt capability

• Double-buffered operation

• Serial clock with programmable polarity and phase

• Control of SPI operation during wait mode

11.3 SPI Block Diagram

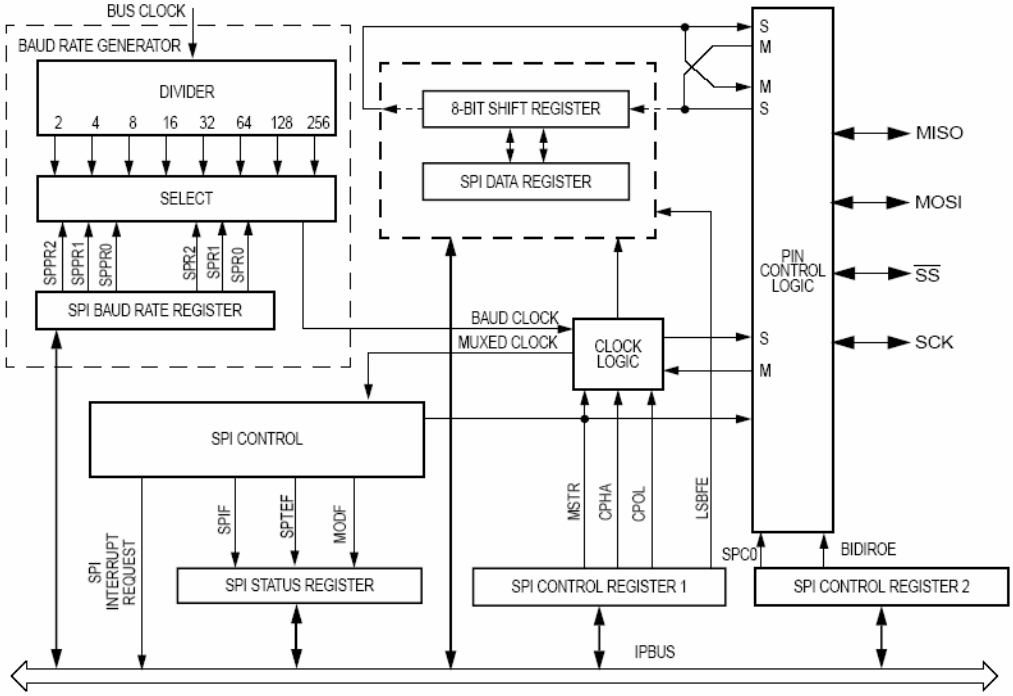


Fig 3.8: SPI Block Diagram

3.11.4 Functional Description

The SPI module allows a duplex, synchronous, serial communication between the MCU and peripheral devices. Software can poll the SPI status flags or SPI operation can be interrupt driven. The SPI system is enabled by setting the SPI enable (SPE) bit in SPI control register 1. While SPE is set, the four associated SPI port pins are dedicated to the SPI function as:

• Slave select (SS)
• Serial clock (SCK)
• Master out/slave in (MOSI)
• Master in/slave out (MISO)

The main element of the SPI system is the SPI data register. The 8-bit data register in the master and the 8-bit data register in the slave are linked by the MOSI and MISO pins to form a distributed 16-bit register. When a data transfer operation is performed, this 16-bit register is serially shifted eight bit positions by the SCK clock from the master; data is exchanged between the master and the slave. Data written to the master SPI data register becomes the output data for the slave, and data read from the master SPI data register after a transfer operation is the input data from the slave. A write to the SPI data register puts data into the transmit buffer if the previous transmission was complete. When a transfer is complete, received data is moved into a receive data register. Data may be read from this double-buffered system any time before the next transfer is complete. This 8-bit data register acts as the SPI receive data register for reads and as the SPI transmit data register for writes. A single SPI register address is used for reading data from the read data buffer and for writing data to the shifter. The clock phase control bit (CPHA) and a clock polarity control bit (CPOL) in the SPI control register 1 select one of four possible clock formats to be used by the SPI system. The CPOL bit simply selects a non-inverted or inverted clock. The CPHA bit is used to accommodate two fundamentally different protocols by shifting the clock by a half cycle or by not shifting the clock. The SPI can be configured to operate as a master or as a slave. When MSTR in SPI control register1 is set, the master mode is selected; when the MSTR bit is clear, the slave mode is selected.

3.12 Motorola Scalable Controller Area Network

3.12.1 Overview

The Motorola Scalable Controller Area Network (MSCAN) definition is based on the MSCAN12 definition which is the specific implementation of the Motorola Scalable CAN concept targeted for the Motorola MC68HC12 Microcontroller Family. The module is a communication controller implementing the CAN 2.0 A/B protocol. The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. MSCAN utilizes an advanced buffer arrangement resulting in a predictable real-time behavior and simplifies the application software.

3.12.2 Features

The basic features of the MSCAN are as follows:

• Implementation of the CAN protocol - Version 2.0A/B

– Standard and extended data frames

– 0 - 8 bytes data length

– Programmable bit rate up to 1 Mbps1

– Support for remote frames

– 5 receive buffers with FIFO storage scheme

• 3 transmit buffers with internal prioritization using a “local priority” concept

• Flexible maskable identifier filter supports two full size extended identifier filters (two 32-bit) or four 16-bit filters or eight 8-bit filters

• Programmable wake-up functionality with integrated low-pass filter

• Programmable loop back mode supports self-test operation

• Programmable listen-only mode for monitoring of CAN bus

• Separate signalling and interrupt capabilities for all CAN receiver and transmitter error states (Warning, Error Passive, Bus-Off)

• Programmable MSCAN clock source either Bus Clock or Oscillator Clock

• Internal timer for time-stamping of received and transmitted messages

• Three low power modes: Sleep, Power Down and MSCAN Enable

3.12.3 MSCAN Block Diagram

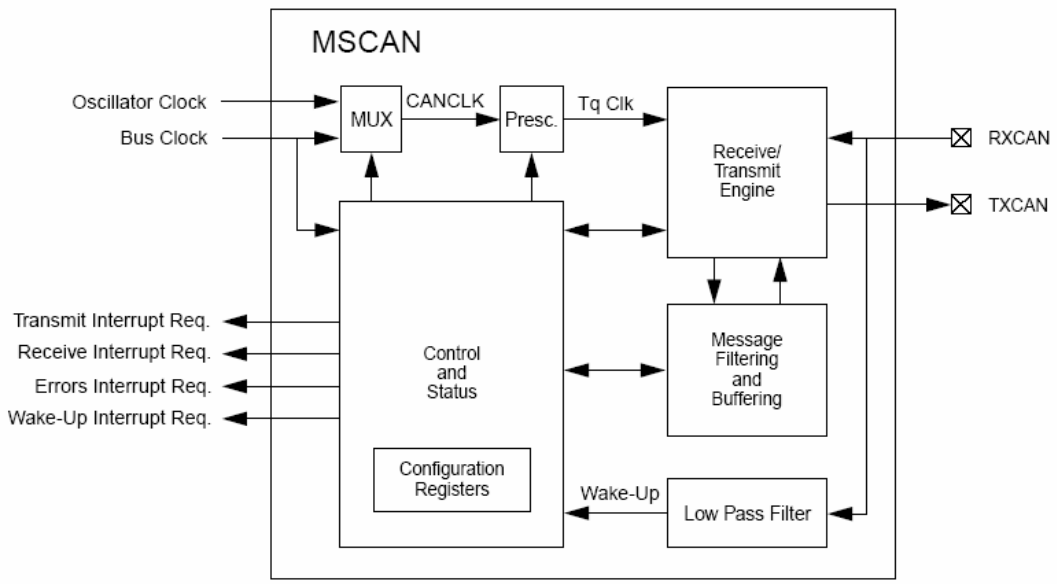


Fig 3.9 MSCAN Block Diagram

The detailed explanation of CAN protocol is given in Chapter 2. The MSCAN operation and the register handling is discussed in Chapter 4.

Chapter 4

Monitoring of Temperature Using temperature Sensor Based on CAN Architecture

4.1 Introduction

Temperature is the most often-measured environmental quantity. This might be expected since most physical, electronic, chemical, mechanical and biological systems are affected by temperature. Some processes work well only within a narrow range of temperatures. Certain chemical reactions, biological processes, and even electronic circuits perform best within limited temperature ranges. When these processes need to be optimized, control systems that keep temperature within specified limits are often used. Temperature sensors provide inputs to those control systems. When temperature limits are exceeded, action must be taken to protect the system. In these systems, temperature sensing helps enhance reliability.

4.2 Architecture of the implemented system

The implemented system is constituted of two CAN nodes, the CAN1 node that is located at the source of temperature and the other is located at where the temperature is to be monitored. The following figure illustrates the block diagram of the architecture of the implemented system.

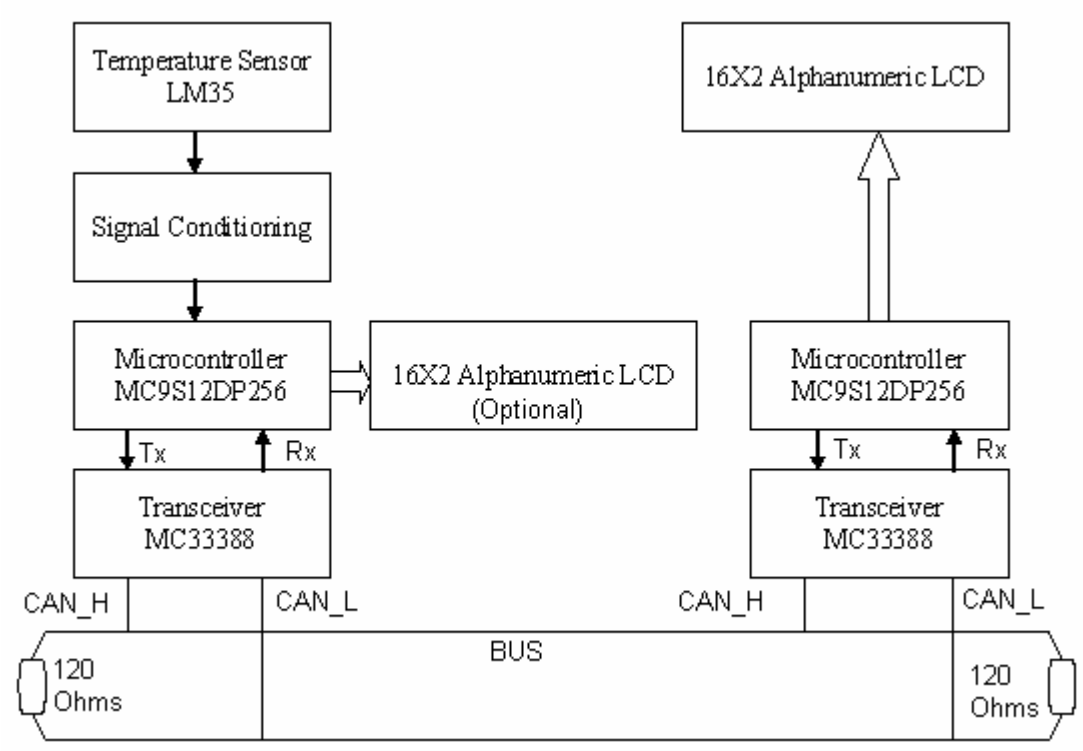


Fig 4.1: Block diagram of the implemented system

As shown in the block diagram each node is formed by a transceiver MC33388 and 16-bit microcontroller MC9S12DP256B. MC33388 is a low speed CAN transceiver and supports

communication speeds up to 125K Bauds. So these two CAN nodes communicate via a coaxial cable with a transmission rate of 250Kbps.

4.3 Description of the implemented system

4.3.1 Temperature Sensor (LM35)

The first stage consists of the conventional sensor of temperature (LM35). The LM35 is a precision integrated-circuit temperature sensor, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. The LM35 temperature sensor is mounted at where the temperature measurement has to be taken. The circuit diagram of LM35 is as shown.

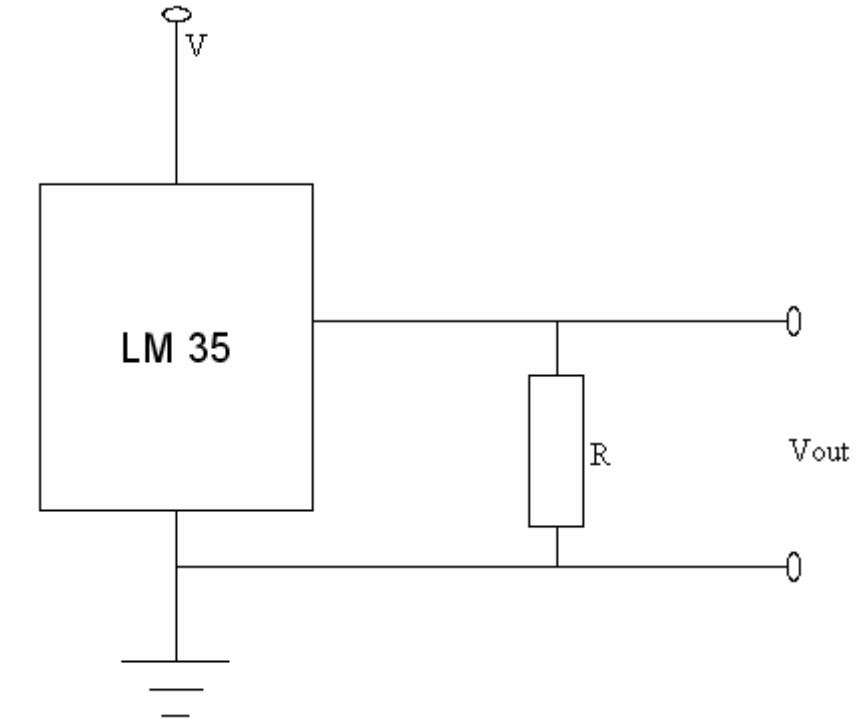


Fig 4.2 The circuit diagram of LM35

The circuit shown is the commonly used one and the supply voltage for the sensor has a wide range from 4V to 30 V. Typically 5V or 12V are used for biasing the LM35, in the present implementation we have used a single 5V supply. The R is calculated as $R = V/10^{-6}$Ohms. Actually R ranges from 80KΩ to 600 KΩ but generally we use 80 K Ω.

The sensor converts the temperature value into voltage signal, and the output of the sensor possesses a very low voltage level, a few milli volts. The output voltage is converted to

temperature by a simple conversion factor. The sensor has a sensitivity of 10 mV/ ºC, i.e. there is a 10 mV raise in the sensor output voltage for each one degree centigrade raise in the temperature. The conversion factor in order to get the temperature value is 100 ºC / V, which is the reciprocal of the sensitivity which is stated above.

The general equation used to convert output voltage to temperature is given by

$$\text{Temperature (ºC)} = V_{out} \text{ X } (100 \text{ ºC / V}) \qquad (4.1)$$

Where V_{out} is the output voltage of the temperature sensor. For example the output voltage is 30mV or .30 V, and then the temperature is 30ºC.

4.3.2 Signal Conditioning circuit

Since the temperature sensor produces a low output voltage signal, there is a need of a signal conditioning circuit in order to transmit the signal to the input of the analog to digital converter. The signal conditioning circuit is designed with 741 operational amplifier. Operational amplifiers can be used as linear amplifiers with external negative feedback. The negative feed back is achieved by a voltage divider circuit which feeds back a fraction of the output signal to the inverting input. As operational amplifiers have a very high open loop gain, very strong negative feed back can be provided. This makes strong use of all of the advantages of negative feed back such as:

- Reduction of distortion,
- Favorable input and output resistances,
- Stable working parameters.

Depending on how (in which form) the negative feed back is achieved and how the signal is fed to the input, different types of amplifiers can be designed. As per our requirement we have designed a signal conditioning amplifier using non inverting configuration. The circuit diagram of the non inverting amplifier is as shown in figure 4.3.

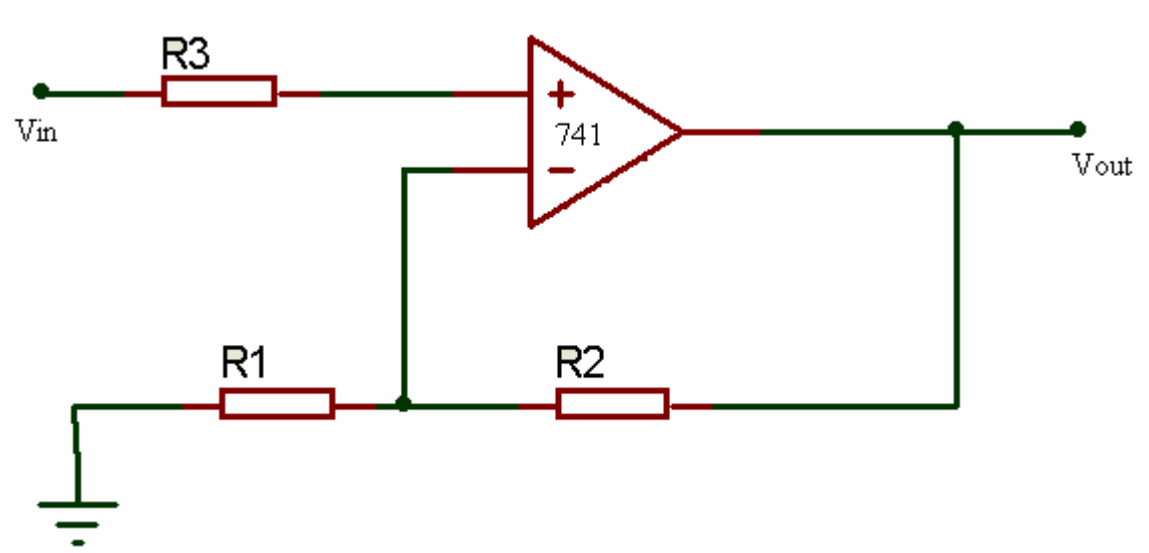


Fig 4.3: Signal conditioning circuit

The non-inverting amplifier feeds the input signal to the non-inverting input. The negative feed back signal is derived from a voltage divider from the output signal and is fed to the inverting input. In the above circuit the negative feed back circuit is built around resistors R1 and R2. The voltage drop across R1 which is a fraction of output voltage is fed back to the inverting input of Op Amp.

The gain of the non inverting amplifier is given by

$$Gain = 1 + \frac{R2}{R1} \tag{4.2}$$

The gain response is independent of R3. So, we choose R3 to be the parallel combination of R1 and R2 so that the effect of the bias currents is reduced.

The output voltage is given by

$$Vout = (1 + \frac{R2}{R1}) * Vin \tag{4.3}$$

The LM 35 output varies from 0V to 1V as the temperature varies from 0ºC to 100ºC. The ATD converter input ranges from 0V to 5V, so in order to interface the temperature sensor to the ATD converter we need an amplifier having a gain of 5. After signal conditioning is done the voltage signal is fed to the ATD converter of MC9S12DP256B microcontroller

4.3.3 Analog to Digital Converter

Analog to digital converter is used to translate the analog signals to digital numbers so that the microcontroller can read the data. The ADC precision is the number of the distinguishable

ADC inputs (e.g., 256 alternatives, 8 bits). The ADC range is the maximum and minimum ADC input (volts, amperes). The ADC resolution is the change in input that causes the digital output to change by 1.

Range (volts) = Precision (alternatives) X Resolution (volts)

The ADC is linear if the resolution is constant through the range. The speed is the time to convert the analog input into digital output.

The microcontroller MC9S12DP256B has built in ATD converter. The flow chart for how to initialize and how to use ATD converter is as shown.

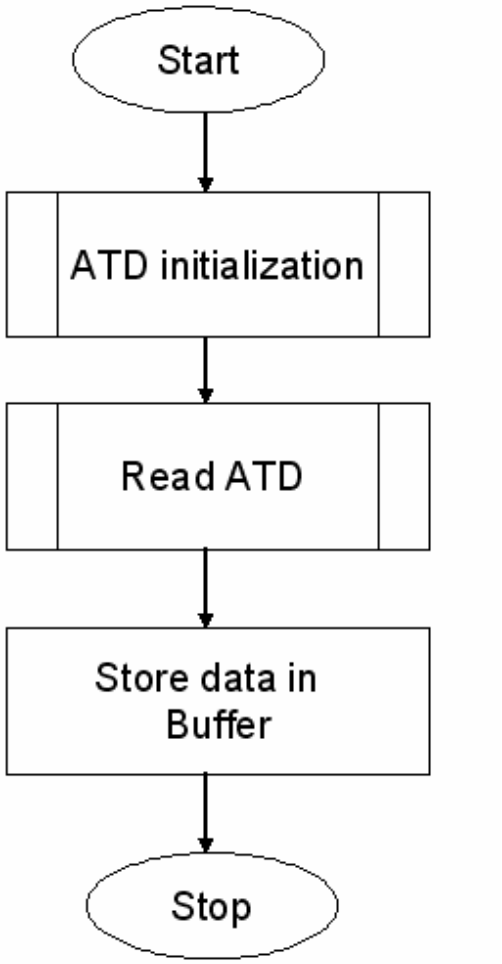
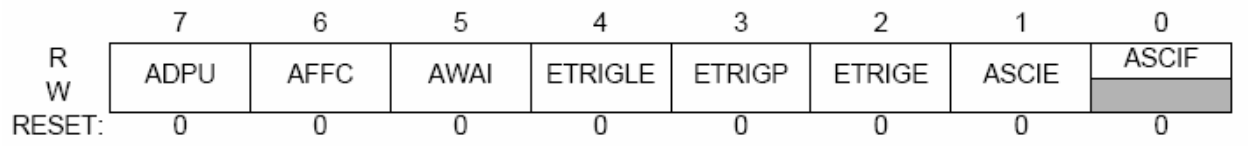


Fig 4.4: ATD converter flow chart

The on chip ATD converter ATD_10B8C is an 8-channel, 10-bit, multiplexed input successive approximation analog to digital converter. Inorder to use the ATD converter we have to follow the procedure as shown in the flow chart. The first sub routine in the flow chart is initialization of ATD. For initialization write the control word 0X80 in the control register ATD0CTL2, this enables the ATD.

The control register ATD0CTL2 is as shown in figure.



	7	6	5	4	3	2	1	0
R W	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
RESET:	0	0	0	0	0	0	0	0

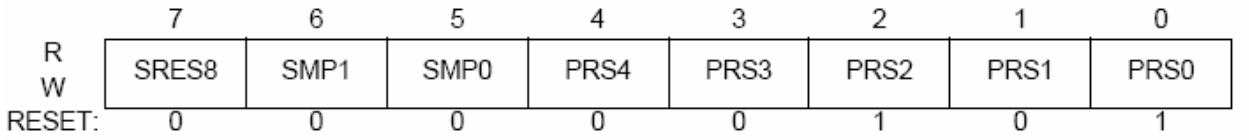
Fig 4.5: Control register ATD0CTL2

The MSB in ATD0CTL2 is ADPU bit, i.e. ATD Power Down. This bit provides on/off control over the ATD_10B8C block allowing reduced MCU power consumption. Because analog electronic is turned off when powered down, the ATD requires a recovery time period after ADPU bit is enabled.

1 = Normal ATD functionality

0 = Power down ATD

After enabling process is done now we have to select the resolution and clock frequency using the control register ATD0CTL4. The control register ATD0CTL4 is as shown in figure.



	7	6	5	4	3	2	1	0
R W	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
RESET:	0	0	0	0	0	1	0	1

Fig 4.6 Control register ATD0CTL4

This register selects the conversion clock frequency and the resolution of the A/D conversion (i.e.: 8-bits or 10-bits). Writes to this register will abort current conversion sequence but will not start a new sequence. The first bit SRES8 selects the ATD resolution. This bit selects the resolution of ATD conversion results as either 8 or 10 bits. The A/D converter has an accuracy of 10 bits; however, if low resolution is required, the conversion can be speeded up by selecting 8-bit resolution.

1 = 8 bit resolution
0 = 10 bit resolution

The control bits PRS4, PRS3, PRS2, PRS1, PRS0 are used for ATD Clock Prescaler. These 5 bits are the binary value prescaler value PRS. The ATD conversion clock frequency is calculated as follows:

$$ATDclock = \frac{[BusClcok]}{[PRS+1]} X\,0.5 \qquad (4.4)$$

Note that the maximum ATD conversion clock frequency is half the Bus Clock. The default (after reset) prescaler value is 5 which results in a default ATD conversion clock frequency that is Bus Clock divided by 12. We have used the prescaler value 5 which gives the ATD clock frequency as 1.33MHz for 16MHz Bus Clock.

After initialization is done we have to perform the read operation. In read ATD subroutine we use the control register ATD0CTL5 which is shown below.

	7	6	5	4	3	2	1	0
R	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
W								
RESET:	0	0	0	0	0	0	0	0

Fig 4.7 Control register ATD0CTL5

The MSB in the control register is DJM bit which is used for Result Register Data Justification

This bit controls justification of conversion data in the result registers.

1 = Right justified data in the result registers

0 = Left justified data in the result registers

We have used right justified data in the result registers.

The SCAN bit is used to choose Continuous Conversion Sequence Mode. This bit selects whether conversion sequences are performed continuously or only once.

1 = Continuous conversion sequences (scan mode)

0 = Single conversion sequence

We have used continues conversion sequences by setting the SCAN bit to 1.

After this we have to choose the analog input channel by using the bits CC, CB, CA. These bits select the analog input channel(s) whose signals are sampled and converted to digital codes. Presently we have selected the analog input 0 by making the CC, CB and CA bits zero.

Now the status register is monitored for conversion complete flags. Which indicates whether the conversion is completed or not. The ATD status register 1(ATD0STAT1) is shown below.

	7	6	5	4	3	2	1	0
R	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
W								
RESET:	0	0	0	0	0	0	0	0

Fig 4.8 Status register ATD0STAT1

This is a read only register, CCFx is the Conversion Complete Flag x (x=7, 6, 5, 4, 3, 2, 1, 0)

A conversion complete flag is set at the end of each conversion in a conversion sequence. The flags are associated with the conversion position in a sequence (and also the result

register number). Therefore, CCF0 is set when the first conversion in a sequence is complete and the result is available in result register ATD0DR0; CCF1 is set when the second conversion in a sequence is complete and the result is available in ATD0DR1, and so forth. A flag CCFx (x=7, 6, 5, 4, 3, 2, 1, 0) is cleared when one of the following occurs:

A) Write to ATD0CTL5 (a new conversion sequence is started)

B) If AFFC=0 and read of ATD0STAT1 followed by read of result register ATD0DRx

C) If AFFC=1 and read of result register ATD0DRx

1 = Conversion number x has completed, result ready in ATD0DRx

0 = Conversion number x not completed

Wait until the conversion flag is set and after that we can get the converted data from ATD conversion result register ATD0DR0. Now the data is stored in a buffer for further processing.

According to the temperature value the ATD converter gives the corresponding digital data. Now this data is to be fed to the on chip CAN of MC9S12DP256B microcontroller and the data package containing the monitored temperature value is transmitted.

4.3.4. MSCAN as Transmitter

The Controller Area Network (the CAN bus) is a serial communications bus for real-time control applications; operates at data rates of up to 1 Megabits per second, and has excellent error detection and confinement capabilities. Data messages transmitted from any node on a CAN bus do not contain addresses of either the transmitting node, or of any intended receiving node. Instead, the content of the message is labeled by an identifier that is unique throughout the network. All other nodes on the network receive the message and each performs an acceptance test on the identifier to determine if the message, and thus its content, is relevant to that particular node. If the message is relevant, it will be processed; otherwise it is ignored. The unique identifier also determines the priority of the message. The lower the numerical value of the identifier, the higher the priority. In situations where two or more nodes attempt to transmit at the same time, a non-destructive arbitration technique guarantees that messages are sent in order of priority and that no messages are lost.

The MC9S12DP256B has five CAN 2.0 A, B software compatible modules. This module supports standard and extended data frames as well as remote frames with a programmable bit rate up to 1Mbps. The MSCAN12 has a triple transmit buffer scheme which allows multiple messages to be sent up in advance and achieve an optimized performance. Received messages are stored in five stage input FIFO. The module has configurable hardware

identifier filter which may be applied to incoming messages. A successful transmission or a message reception with a matching identifier will be flagged.

The ATD converted temperature value is transmitted to the CAN node2 through the CAN bus using the MSCAN12 module. The following flow chart describes how to initialize and use the MSCAN module in conjunction with ATD converter.

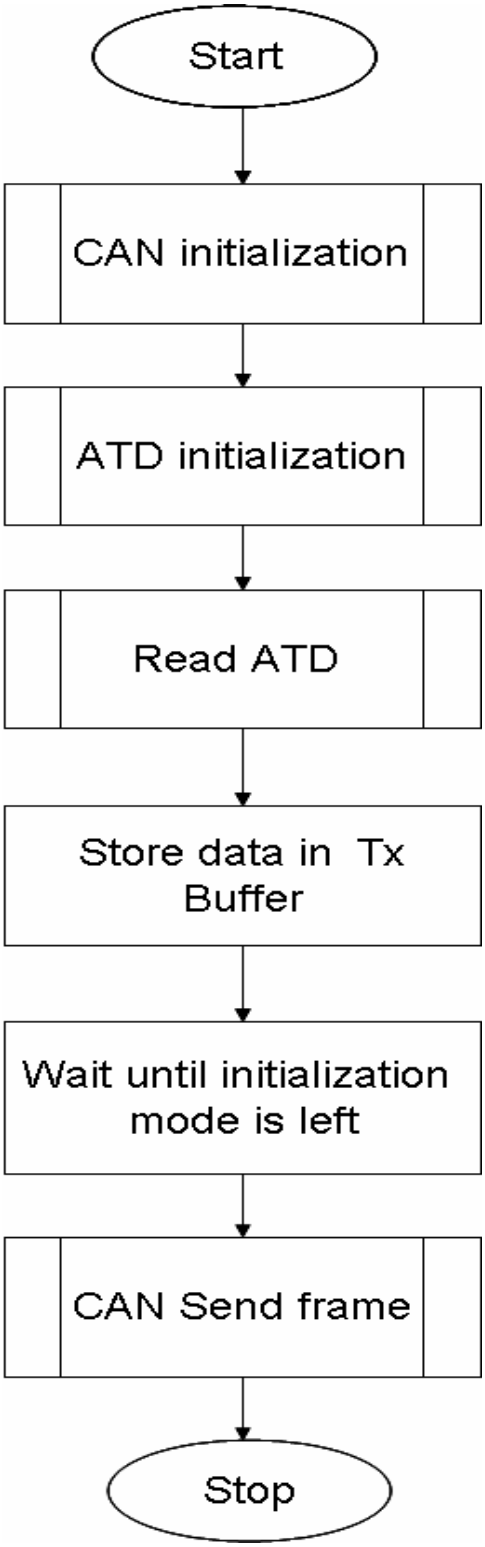


Fig 4.9: Flow chart for CAN transmission

The ATD subroutines are explained in the earlier section. In CAN initialization subroutine the registers CAN0CTL0, CAN0CTL1, CAN0BTR0 and CAN0BTR1 are used. The control register CAN0CTL0 is as shown

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
Write:								
Reset:	0	0	0	0	0	0	0	1

4.10 Control register CAN0CTL0

The LSB is INITRQ (Initialization Mode Request). When this bit is set by the CPU, the MSCAN skips to Initialization Mode. Any ongoing transmission or reception is aborted and synchronization to the bus is lost. The module indicates entry to Initialization Mode by setting INITAK=1. Writing to other bits in CAN0CTL0, CAN0RFLG, CAN0RIER, CAN0TFLG or CAN0TIER must only be done after Initialization Mode is left, which is INITRQ=0 and INITAK=0.

1 = MSCAN in Initialization Mode.

0 = Normal operation.

The software waits in CAN initialization subroutine until the initialization mode is left.

Now the MSCAN is enabled using the control register CAN0CTL1. The control register CAN0CTL1 is as shown.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
Write:								
Reset:	0	0	0	1	0	0	0	1

4.11 Control register CAN0CTL1

The MSB is CANE (MSCAN Enable). The MSCAN module is enabled if CANE bit is 1 and is disabled if it is 0.

By setting the control word 0X80 in CAN0CTL1 the MSCAN module is enabled.

MSCAN bus timing registers CAN0BTR0 and CAN0BTR1 provides for various bus timing control of the MSCAN module. The bus timing register CAN0BTR0 is given below.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Write:								
Reset:	0	0	0	0	0	0	0	0

4.12 Control register CAN0BTR0

The synchronization jump width defines the maximum number of time quanta (Tq) clock cycles a bit can be shortened or lengthened to achieve resynchronization to data transitions on the bus. SJW1, SJW0 are the Synchronization Jump Width bits. On writing SJW1 = 1 and SJW0 = 1 the synchronization jump width is selected as 4 Tq clock cycles.

The Baud Rate Prescaler bits BRP [5-0] determine the time quanta (Tq) clock which is used to build up the individual bit timing. The prescaler value of 8 is obtained by setting the BPR0, BPR1 and BPR2 to 1 and remaining bits BPR3, BPR4 and BPR5 to 0.

The bus timing register CAN0BTR1 is as shown.

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10

4.13 Bit timing register CAN0BTR1

Time Segment 2 (TSEG22 – TSEG20)

Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point.

Time Segment 1 (TSEG13 – TSEG10)

Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point.

By writing 0X3A, time segment 2 (TSEG2) values are programmed as 4 Tq clock cycles and the time segment 1 (TSEG1) values are programmed as 11 Tq clock cycles.

The bit time is determined by the oscillator frequency, the baud rate prescaler, and the number of time quanta (Tq) clock cycles per bit.

$$\text{Bit Time} = \frac{(\text{Prescaler value})}{f_{CANCLK}} \bullet (1 + \text{TimeSegment1} + \text{TimeSegment2}) \qquad (4.5)$$

Next step is to exit the initialization mode by writing 0 to CAN0CTL0 and wait for normal mode.

After the initialization of MSCAN, ATD converter is initialized and the input analog voltage signal (which is the output of temperature sensor) is digitized and is store in a buffer. Now check whether the initialization mode is left or not, then the CAN send frame subroutine is called.

First of all check whether the transmit buffer full or empty by using the CAN0TFLG register. The CAN0TFLG is as shown.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	TXE2	TXE1	TXE0
Write:								
Reset:	0	0	0	0	0	1	1	1

Fig 4.14 Transmitter flag register CAN0TFLG

Transmitter Buffer Empty is indicated by the bits TXE2 - TXE0. This flag indicates that the associated transmit message buffer is empty, and thus not scheduled for transmission. The CPU must clear the flag after a message is set up in the transmit buffer and is due for transmission. The MSCAN sets the flag after the message is sent successfully. Read and write accesses to the transmit buffer will be blocked, if the corresponding TXEx bit is cleared (TXEx='0') and the buffer is scheduled for transmission.

1 = The associated message buffer is empty (not scheduled).

0 = The associated message buffer is full (loaded with a message due for transmission).

The CAN0TBSEL register allows the selection of the actual transmit message buffer. The CAN0TBSEL is as shown in fig.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	TX2	TX1	TX0
Write:								
Reset:	0	0	0	0	0	0	0	0

Fig 4.15 Transmit buffer select register CAN0TBSEL

The bits TX2 - TX0 are used for selecting the transmit buffer. The lowest numbered bit places the respective transmit buffer in the CANTXFG register space (e.g. TX1=1 and TX0=1 selects transmit buffer TX0, TX1=1 and TX0=0 selects transmit buffer TX1) Read and write accesses to the selected transmit buffer will be blocked, if the corresponding TXEx bit is cleared and the buffer is scheduled for transmission.

1 = The associated message Buffer is selected, if lowest numbered bit.

0 = The associated message buffer is deselected.

Select the lowest empty buffer by copying the contents of CAN0TFLG to CAN0TBSEL.

Now the data is stored in Data Segment Registers (DSR0-7). There are eight data segment registers, each with bits DB7-DB0, contain the data to be transmitted or received. The number of bytes to be transmitted or received is determined by the data length code in the

corresponding DLR register. The DLC3 - DLC0 — Data Length Code bits The data length code contains the number of bytes (data byte count) of the respective message. During the transmission of a remote frame, the data length code is transmitted as programmed while the number of transmitted data bytes is always 0. The data byte count ranges from 0 to 8 for a data frame.

We can assign the priority to the transmit buffer by using Transmit Buffer Priority Register (TBPR). This register defines the local priority of the associated message buffer. The local priority is used for the internal prioritization process of the MSCAN and is defined to be highest for the smallest binary number. The MSCAN implements the following internal prioritization mechanisms:

All transmission buffers with a cleared TXEx flag participate in the prioritization immediately before the SOF (Start of Frame) is sent.

The transmission buffer with the lowest local priority field wins the prioritization. In cases of more than one buffer having the same lowest priority, the message buffer with the lower index number wins.

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
Reset:	0	0	0	0	0	0	0	0

Fig 4.16 Transmit buffer priority register CAN0TXTBPR

To achieve highest priority the CAN0TXTBPR is loaded with 0X00. Then the data is sent through the CAN bus with the help of a CAN transceiver.

4.3.5 Fault Tolerant CAN Interface (MC33388)

Each CAN Node is connected physically to the CAN bus line through a transceiver chip MC33388. The transceiver is capable of driving the large current needed for the CAN bus, and has current protection against defected CAN or defected stations. The MC33388 is a low speed CAN fault tolerant physical interface designed for automotive multiplexed electronic systems. The MC33388 addresses the low speed body electronics application, in which the speed of communications is between 10 and 125kBauds, on two wires bus configurations. It is designed to operate in the harsh automotive environment.

The device configuration is described in minimum application schematic is given in figure below. The device is used as CAN transceiver only and other features are not used. The device EN and STB input pins must be connected to 5V in order to set the device in normal

mode. INH and NERR can be left open. WAKE should not be left open and must be connected to a known state, i.e. GND.

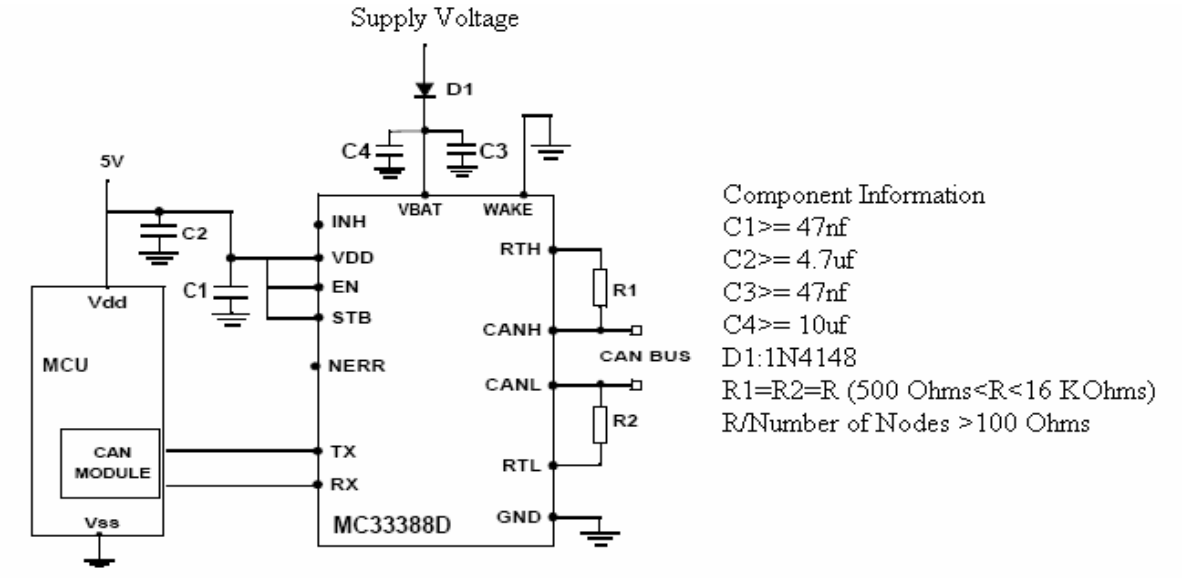


Fig 4.17 MC33388 circuit diagram

The MC33388 is connected as per the schematic shown above. Similar type of connections are made at the receiving node also. Now the data can be transmitted to the receiving CAN node through CAN bus.

4.3.6 MSCAN as Receiver

The implemented CAN receiver is described by the following flow chart

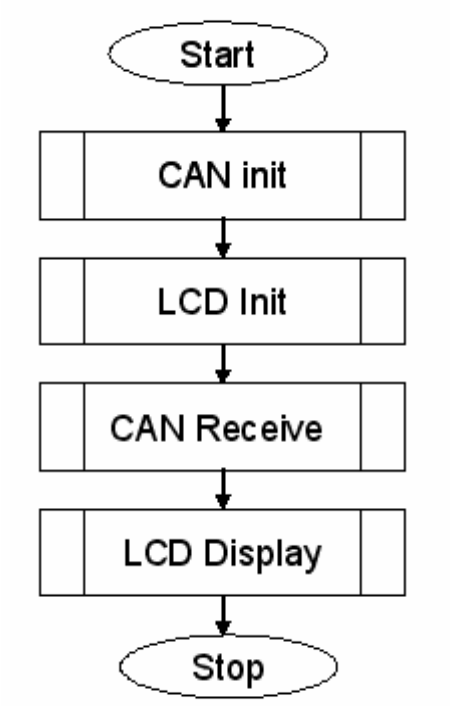


Fig 4.18 CAN receiver flow chart

The initialization of MSCAN is explained in the previous section. Now we have to extract the received data using the CAN receive subroutine. In this first of all we have to find out the data length using CAN0RXDLR register. Now as per the data length, extract that many number of received data bytes from CAN0RXDSR. In order to display the received data LCD display subroutine is called.

4.3.7 LCD display

Seven segment displays (LCD or LED) can be used to display numbers. But these are inadequate to display all the alphabets. So by using an alphanumeric LCD display we can fulfill this requirement. A 2X16 characters LCD is used to display the messages indicating the received or transmitted temperature and to display the temperature values received by the CAN receiver after processing.

The LCD interfacing connections are shown.

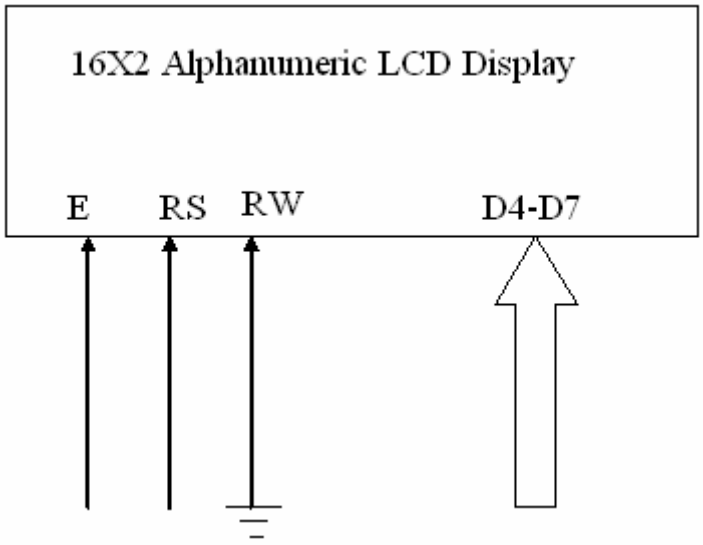


Fig 4.19 LCD connection diagram

The LCD is used only in write mode by grounding the pin RW. To write any command RS pin is made low and for writing a data this pin is made high. If a data word is to be written, data byte is placed on the data bus, register select is made high and E clock is triggered. Similarly to write an instruction, instruction byte is placed on the data bus, register select is made low and E clock is triggered.

A LCD can be used in two ways. One is in 8-bit mode and the other is in 4-bit mode. In 8-bit mode of operation the data byte or instruction is written using 8 I/O lines where as in the 4-bit mode of operation the higher nibble of the data byte is written first followed by the lower nibble. So by using the 4-bit mode we can reduce the number I/O lines required.

We have used 4-bit interface and the LCD is connected to the PORTA of MC9S12DP256B. The upper nibble of PORTA is connected to the data lines D4-D7 of LCD, D3 is connected to enable pin and the D2 pin is connected to the RS pin of LCD Display.

The flow chart of LCD operation is given in fig 4.20.

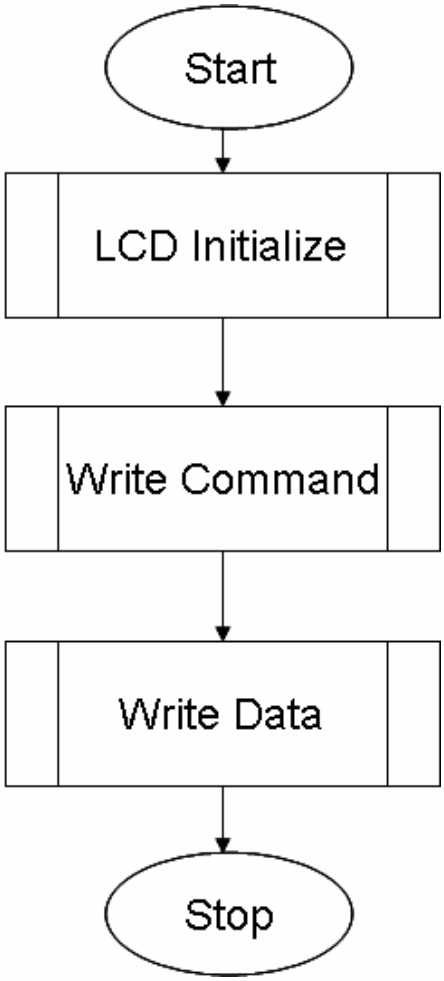


Fig 4.20 LCD flow chart

Hence the temperature data received by the CAN receiver node is displayed using the 16X2 alphanumeric LCD display. The LCD display used in the experiment is as shown.

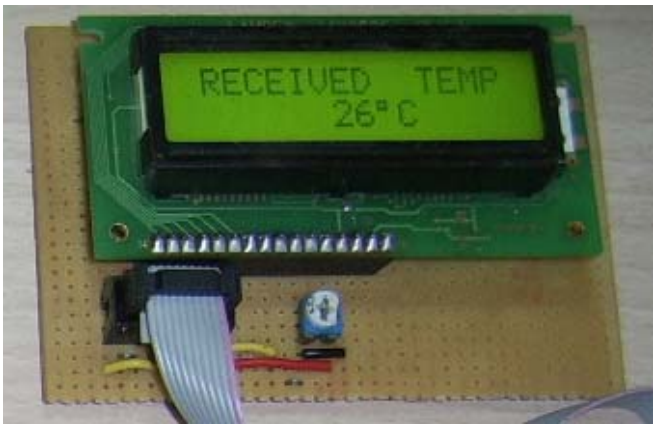


Fig 4.21 LCD Displaying Temperature

4.4.Temperature Controller

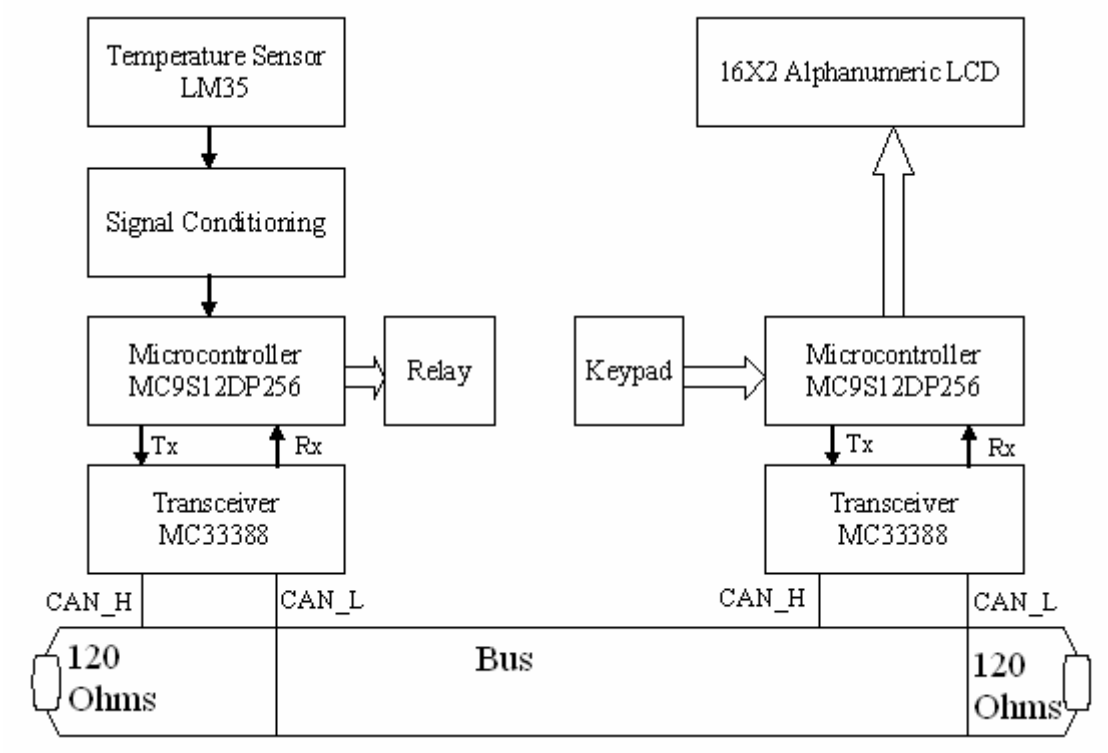


Fig 4.22 Block diagram of the Temperature Controller

The temperature controller system is as shown above. This is as same as temperature monitoring system with slight modifications. The modifications are Node 1 is incorporated with an electromagnetic relay and Node 2 is provided with a keypad. In this system Node 1 and Node 2 acts as transmitter as well as receiver. Node 2 facilitates the user to enter the maximum operating temperature of the device using the keypad. The keypad consists of 3 keys in which two keys are used to enter the temperature value and the third key is used as Enter key. The maximum temperature value entered is transmitted to the CAN Node1 and stored in a buffer. Node 1 compares this value with the monitored temperature and takes the decision to turn on or to turn off the device. The relay is operated according to the decision taken by the Node 1. Hence the device is controlled using CAN networking.

4.5 Experimental Setup

4.5.1 Temperature Monitoring: Transmitter Node (CAN Node1)

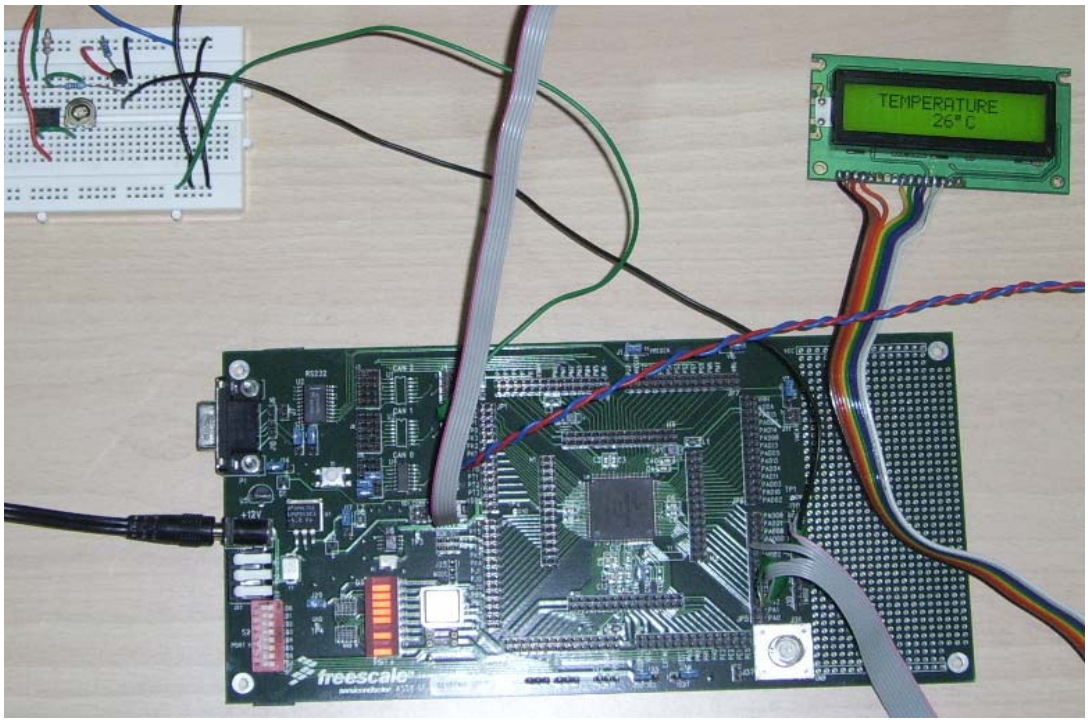


Fig 4.23 Temperature Monitoring: CAN transmitter node

4.5.2 Temperature Monitoring: Receiver Node (CAN Node2)

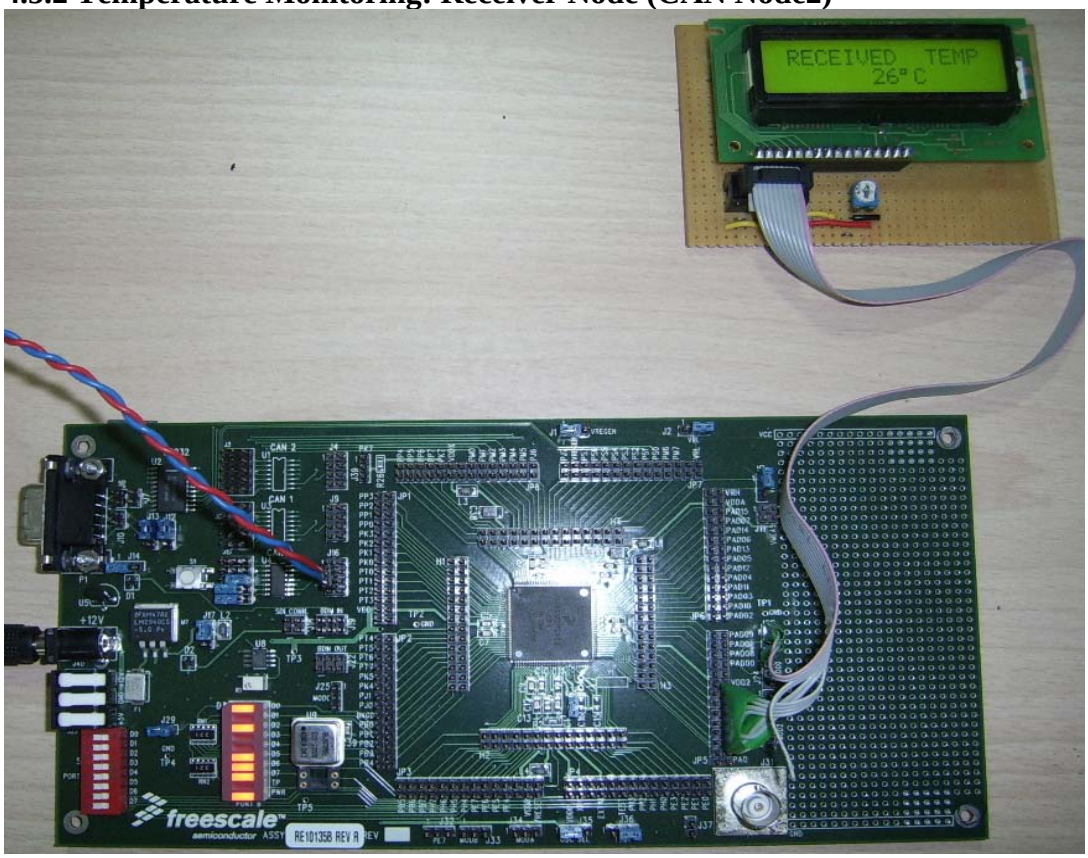


Fig 4.24 Temperature Monitoring: CAN Receiver node

4.5.3 Temperature Controller: CAN Node1

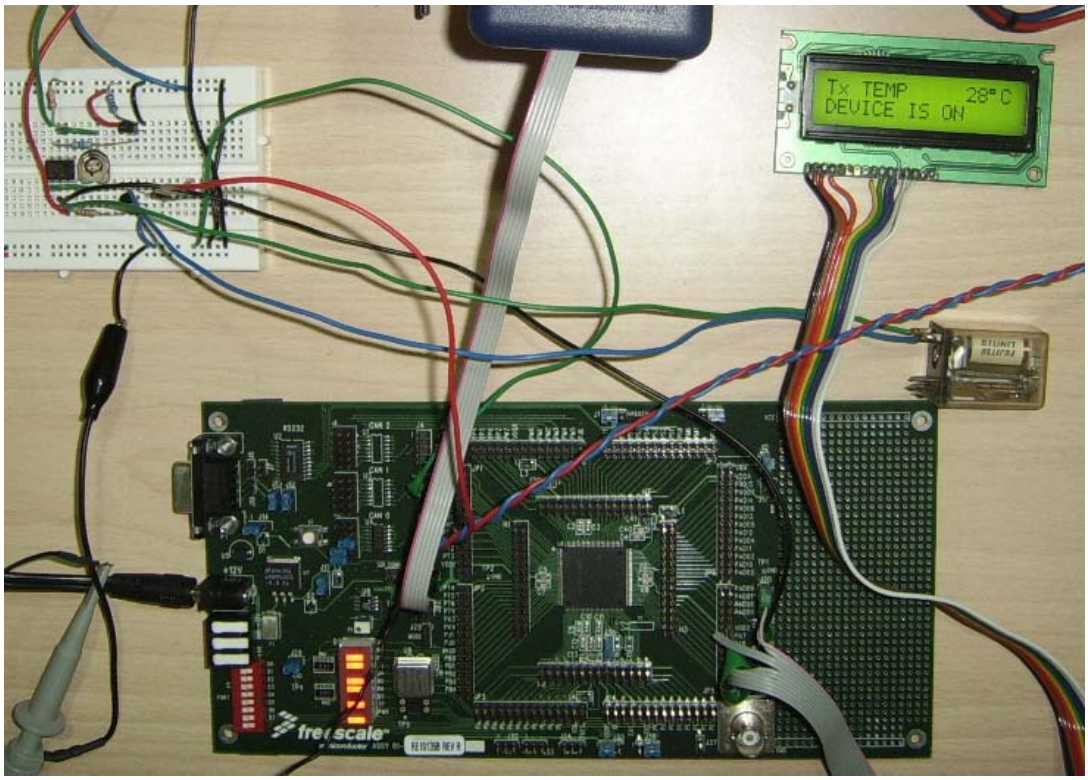


Fig 4.25 Temperature controller: CAN Node 1

4.5.4 Temperature Controller: CAN Node2

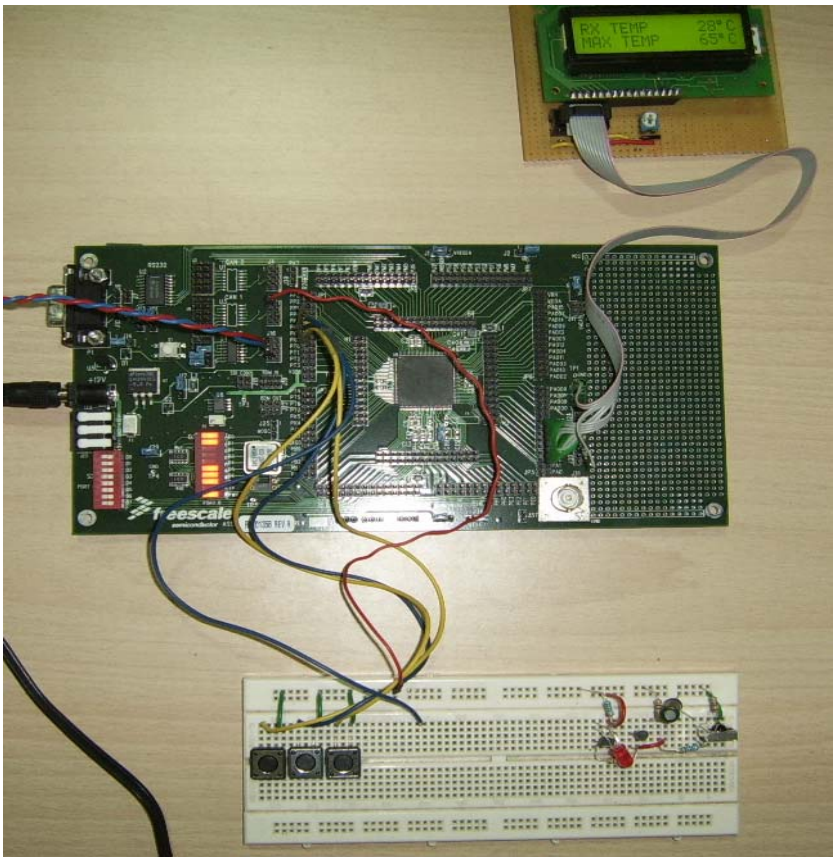


Fig 4.26 Temperature Controller: CAN Node 2

4.6 CAN Frames

4.6.1 Temperature Monitoring: CAN Frames of transmitter node

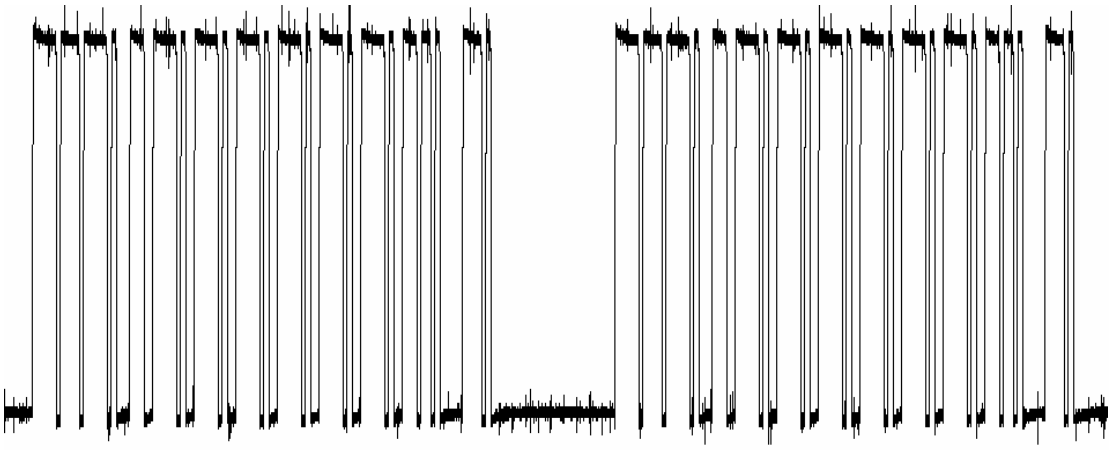


Fig 4.27 CAN Frames of Temperature Monitoring

4.6.2 Temperature Controller: CAN Frame of Node 1

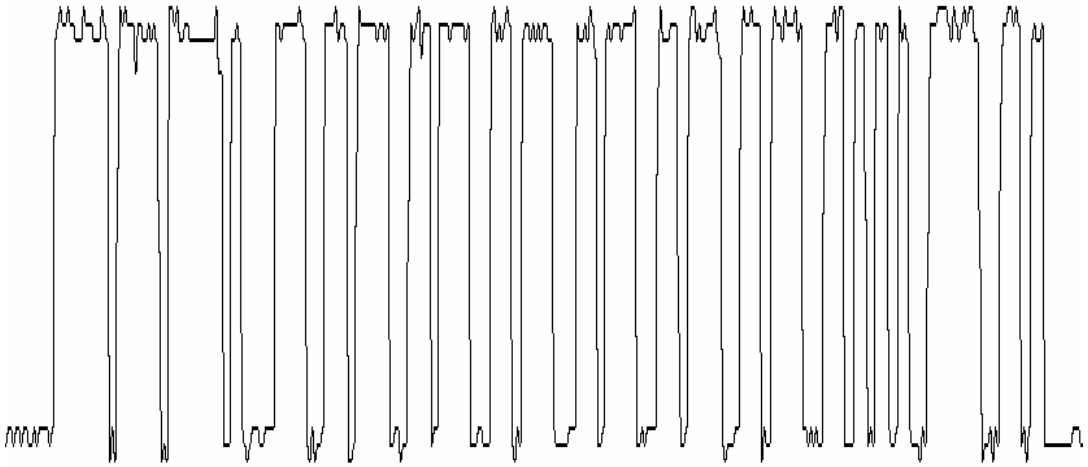


Fig 4.28 CAN Node 1 frame of Temperature Controller

4.6.3 Temperature Controller: CAN Frame of Node 2

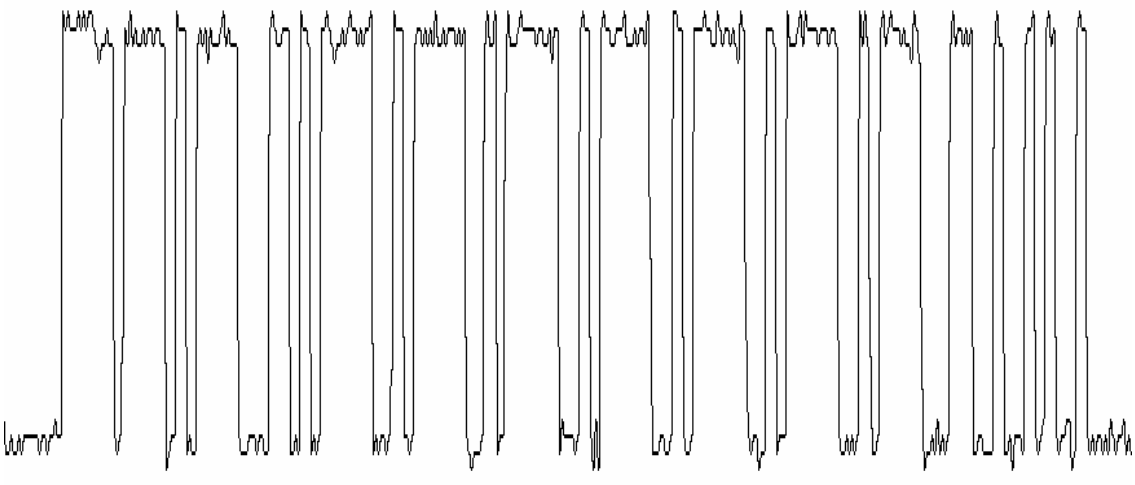


Fig 4.29 CAN Node 2 frame of Temperature Controller

Chapter 5

CONCLUSION AND FUTURE WORK

This thesis is concerned with design techniques for implementation of CAN nodes for data monitoring and taking appropriate decision based on data in the control system. Implementation of CAN for temperature monitoring is successful and the same idea can be applicable to monitor Tire pressure monitoring system, Adaptive Cruise control, power window and Engine management systems in Automotives. This leads to decentralization of control system in vehicles. This can be extended to industrial control vertical, mainly in decentralizing the PLC (programmable Logic Controllers) control mechanisms.

Using Freescale 68HCS12DP256 microcontroller to implement CAN nodes, which contains inbuilt CAN IP core. In chapter I we discussed about definition of embedded systems, networks for embedded systems and features of communication protocols for embedded systems. Chapter II deals with Controller Area Network (CAN) specifications and standards, different types of CAN, its properties, its efficiency, its advantages and disadvantages. Finally we will discuss about Motorola Scalable CAN protocol. Chapter III discusses about architecture, features and tools required for MC9S12DP256 microcontroller. It also discusses about peripherals its supports in brief, background debug mode for debugging, hardware design of circuit board used, Finally, the chapter IV discusses about implementation of temperature monitoring and temperature controlling using CAN architecture including CAN transceiver device MC33388D.

Future work

There is potential future work implementing CAN for Industrial control systems like PLC's etc. This thesis is concerned to temperature monitoring, this can be extended to four nodes, eight nodes and 16 nodes CAN network for different automotive applications. For further decentralization of control systems, sub-network for CAN called LIN (Local Interconnect Network) can be implemented. It is intended in future works, to increase the number of CAN nodes on the bus, using CAN IP's based on FPGA, Signal processors and other microcontrollers. We can select TCP/IP interface to design web based control system and further develop the conversion algorithms to convert CAN data frames to TCP/IP format, for the successful design of Web based control systems.

REFERENCES

[1] Jadsonlee da Silva Sa, Jaidilson Jo da Silva, Miguel Goncalves Wanzeller and Jose Sergio da Rocha Neto, "Monitoring of Temperature Using Smart Sensors Based on CAN Architecture." Proceedings of the 15th International Conference on Electronics, Communications and Computers, 2005 IEEE.

[2] H. F. Othman, Y. R. Aji, F. T. Fakhreddin, A. R. Al-Ali,"Controller Area Networks: Evolution and Applications", 2006 IEEE.

[3] Robert Bosch GmbH, "CAN Specification", Version 2.0, September 1991.

[4] K. Pazul, "Controller Area Network (CAN) basics", AN713, Microchip Technology, USA, Inc, 1999.

[5] P. Richards, "A CAN physical layer discussion", AN228, Microchip Technology, USA, Inc, 2002.

[6] Stuart Robb, East Kilbride, "CAN Bit Timing Requirements", AN1798, Freescale Semiconductor, Inc., Scotland, 2004.

[7] Freescale Semiconductor, "MC9S12DP256B Device User Guide V02.15", Freescale Semiconductor, Inc. Jan 11, 2005.

[8] Freescale Semiconductor, "MSCAN Block Guide V02.15", Freescale Semiconductor, Inc. 15 JUL 2004.

[9] National Semiconductor, "LM35 Precision Centigrade Temperature Sensors- Data sheet", National Semiconductor Corporation, November 2000.

[10] National Semiconductor, "Temperature Sensor Handbook", National Semiconductor Corporation, 2000.

[11] Jonathan W. Valvano. "Embedded Microcomputer Systems". Singapore: Thomson Brooks / Cole, 2002.

[12] Todd D. Morton. “Embedded Microcontrollers”. Delhi: Pearson Education, 2001

[13] Raj Kamal. “Embedded Systems”. New Delhi: Tata McGraw-Hill Publishing Company Limited, 2005.

[14] Wayne Wolf. “Components as Components”. New Delhi: Morgan Kaufmann Publishers, 2001.

[15] Steve heath. “Embedded Systems Design”. New Delhi: Published by Elsevier

[16] “CAN Applications fields”, http://www.can-cia.org/applications/, Dec.2005

[17] “Controller Area Network – CAN Information”, http://hem.bredband.net/stafni/developer /frames.htm , 2007.

[18] Industrial Controller Area Network (CAN) Applications. http://www.freescale.com/webapp/sps/site/application.jsp?nodeId=02430ZNtdx4J11

[19] www.freescale.com